USER MANUAL

CDS-3310

Manual Rev. 1.0f

By Galil Motion Control, Inc.

Galil Motion Control, Inc.
270 Technology Way
Rocklin, California 95765
Phone: (916) 626-0101
Fax: (916) 626-0102
E-mail Address: support@galilmc.com
URL: www.galilmc.com

Rev 01/10

Using This Manual

This user manual provides information for proper operation of the CDS-3310 controller. A separate manual, the Command Reference, contains a description of the commands available for use with this controller.

Users of a 1-4 axis system should note that axes are labeled XYZW.

Users of a 1-8 axis system should note that axes are labeled A,B,C,D,E,F,G,H. The axes A,B,C,D may be used interchangeably with X, Y, Z, W.

WARNING: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machine. Galil shall not be liable or responsible for any incidental or consequential damages.

ContentsContents i

Chapter 1 Overview 1

Introduction 1
Amplifier Specifications 2
CDS-3310 Functional Elements 3
- Microcomputer Section 3
- Motor Interface 3
- Communication 3
- General I/O 3
- System Elements 4
- Motor 4
- Amplifier (Driver) 4
- Encoder 4
- Watch Dog Timer 5

Chapter 2 Getting Started 6

CDS-3310 Layout and Dimensions 7
Recommended Components 7
Installing the CDS-3310 8
- Step 1. Install Jumpers on the CDS-3310 8
- Step 2. Connect 18 to 72 VDC Power to the Controller 9
- Step 3. Install Windows Communication Software 9
- Step 4. Establish Communication 10
- Step 5. Make Connections to Encoder 11
- Step 6. Configure Amplifier 11
- Step 7a. Connect Brush Servo Motor 12
- Step 7b. Connect Brushless Servo Motor 12
- Step 8. Close the Loop 13
- Step 9. Tune the Servo System 14

Design Examples 15
- System Set-up 15
- Profiled Move 15
- Position Interrogation 15
- Absolute Position 15
- Velocity Control 16
- Operation Under Torque Limit 16
- Interrogation 16
- Motion Programs with Loops 16
- Control Variables 17

Chapter 3 Connecting I/O 19

Overview 19
Inputs 19
- General Use Digital Inputs 19
- Limit Switch Inputs 19
- Home Switch Input 20
- Abort Input 21
- Reset Input 21
- Auxiliary Encoder Inputs 21
- Analog Inputs 22

Outputs 22

General Use Digital Outputs....22
Amp Enable....22
Output Compare....22
Error Output....22
Brake Output....23
Analog Output....23

Chapter 4 Communication 25

Introduction....25
RS232 Port....25
Ethernet Configuration....25
Communication Protocols....25
Addressing....25
Communicating with Multiple Devices (Handles)....26
Modbus....27
Multicasting....28
Using Third Party Software....28
Data Record....29
Data Record Map....29
Explanation of Status and Axis Switch Information....32
Notes Regarding Velocity and Torque Information....33
QZ Command....33
Controller Response to Commands....33
Unsolicited Messages Generated by Controller....33
Galil Software Tools and Libraries....34

Chapter 5 Command Basics 35

Introduction....35
Command Syntax - ASCII....35
Command Syntax - Binary....37
Binary Command Format....37
Binary command table....38
Controller Response to DATA....39
Interrogating the Controller....39
Interrogation Commands....39
Summary of Interrogation Commands....40
Interrogating Current Commanded Values....40
Operands....40

Chapter 6 Programming Motion 41

Overview....41
Independent Axis Positioning....42
Command Summary - Independent Axis....42
Operand Summary - Independent Axis....43
Examples....43
Independent Jogging....44
Command Summary - Jogging....45
Operand Summary - Independent Axis....45
Examples....45
Position Tracking....46
Example Motion 1:....47
Example Motion 2:....47
Example Motion 3:....48
Trip Points....49

Command Summary – Position Tracking Mode 49
Electronic Gearing 50
Ramped Gearing 50
Example – Electronic Gearing Over a Specified Interval 51
Command Summary - Electronic Gearing 52
Electronic Cam 53
Contour Mode 4
Specifying Contour Segments 4
Additional Commands 5
Command Summary - Contour Mode 6
General Velocity Profiles 6
Example 6
Virtual Axis 9
Dual Loop (Auxiliary Encoder) 9
Additional Commands for the Auxiliary Encoder 9
Backlash Compensation 10
Example 10
Motion Smoothing 11
Using the IT Command: 11
Example 12
Homing 13
Example 13
Command Summary - Homing Operation 15
Operand Summary - Homing Operation 15
High Speed Position Capture (The Latch Function) 15
Example 15

Chapter 7 Application Programming 16

Overview 16
Using the DOS Editor to Enter Programs 16
Edit Mode Commands 17
Example 17
Program Format 18
Using Labels in Programs 18
Special Labels 18
Commenting Programs 19
Executing Programs - Multitasking 19
Debugging Programs 20
Trace Command 20
Error Code Command 21
Stop Code Command 21
RAM Memory Interrogation Commands 21
Operands 21
Example 22
Program Flow Commands 22
Event Triggers & Trippoints 22
Conditional Jumps 26
IF, ELSE, AND ENDIF 28
Subroutines 29
Stack Manipulation 30
Auto-Start Routine 30
Automatic Subroutines for Monitoring Conditions 30
Mathematical and Functional Expressions 35
Mathematical Operators 35
Bit-Wise Operators 35

Functions 36
Variables 37
Programmable Variables 37
Operands 38
Common Operands 38
Arrays 39
Defining Arrays 39
Assignment of Array Entries 39
Uploading and Downloading Arrays to On Board Memory 40
Automatic Data Capture into Arrays 40
Deallocating Array Space 42
Input of Data (Numeric and String—Serial Port Only) 42
Input of Data 42
Operator Data Entry Mode 43
Communication Interrupt 43
String Variables 45
Output of Data (Numeric and String) 45
Sending Messages 45
Displaying Variables and Arrays 46
Interrogation Commands 47
Formatting Variables and Array Elements 48
Converting to User Units 49
Hardware I/O 50
Digital Outputs 50
Digital Inputs 51
Analog Inputs 51
Analog Output 52
The Auxiliary Encoder Inputs 52
Input Interrupt Function 53
Extended I/O of the CDS-3310 Controller 53
Configuring the I/O of the CDS-3310 54
Saving the State of the Outputs in Non-Volatile Memory 54
Accessing Extended I/O 54
Example Applications 55
Wire Cutter 55
Backlash Compensation by Sampled Dual-Loop 56

Chapter 8 Hardware & Software Protection 58

Introduction 58
Hardware Protection 58
Output Protection Lines 58
Input Protection Lines 59
Amplifier Status LED's 59
Software Protection 59
Programmable Position Limits 60
Off-On-Error 60
Automatic Error Routine 60
Limit Switch Routine 61
Amplifier Error Routine 62

Chapter 9 Troubleshooting 64

Overview 64
Installation 64
Communication 65

Stability 65
Operation 65

Chapter 10 Theory of Operation 66

Overview 66
Operation of Closed-Loop Systems 68
System Modeling 69
Motor-Amplifier 70
Encoder 70
DAC 70
Digital Filter 70
ZOH 72
System Analysis 72
System Design and Compensation 74
The Analytical Method 74

Chapter 11 Distributed Control 78

Overview 78
Configuring Distributed Control 78
Configuration Example 79
Manual Slave IP Configuration 79
Global versus Local Commands 80
Global Commands 80
Local Commands 81
Accessing the I/O of the Slaves 81

Appendices 82

Electrical Specifications 82
Servo Control 82
Input / Output 82
Power 82
Performance Specifications 83
Connectors for CDS-3310 84
J1 MOTOR Output 4-pin AMP Universal Mate-N-Lok 84
J2 ENCODER 15-PIN Hi-density Female D-sub 84
J3 I/O 37-pin Female D-sub 84
J4 RS-232 Serial Port 85
J5 POWER 2-pin AMP Universal Mate-N-Lok 85
J8 100Base-T Ethernet 85
Mating Connectors 85
Pin-Out Description for CDS-3310 86
Jumper Description for CDS-3310 87
Accessories and Options 87
ICM-3300 89
Opto Isolation Settings 89
Input Isolation 90
Output Isolation 91
Screw Terminal Description 92
SR-19900 93
Introduction 93
Layout 94
Pinout 94
Configuration 94
DB-28040 95

Introduction 95
Electrical Specifications 95
16 bit Analog Inputs 95
Interfacing to the Digital I/O 95
Using Opto-22 G4 Series Racks with DB-28040 95
Pinout 96
Layout 97
CABLE-15-1m 98
Communicating with OPTO-22 SNAP-B3000-ENET 98
List of Other Publications 100
Training Seminars 100
Contacting Us 101
WARRANTY 102

Index **103**

Chapter 1 Overview

Introduction

The Galil CDS-3310 controller/drive system is a single-axis integrated Ethernet controller and brush/brushless amplifier (18-72 VDC, 7 A continuous, 10 A peak) that can be used in one of several ways, depending on how many total machine axes need to be controlled:

Machine Axes	Mode	Communication Commands
1	The CDS-3310 is used by itself for single axis applications	n/a
1.5	The CDS-3310 is used by itself in conjunction with its second encoder input for gearing, backlash compensation, or reading a hand wheel.	n/a
2 to 8	The CDS-3310 is used in conjunction with up to 7 other CDS-3310s. In this "distributed" mode of operation, programming is simplified because multiple controllers behave as a single multi-axis controller. Only the master controller requires an application program and/or the host PC communicates only with the master controller.	HA, HC (see Chapter 11)
2 or more	The CDS-3310 is used in conjunction with one or more Galil DMC-14x5, 21x2/3, 22x0 or CDS-3310 Ethernet controllers. In this mode of operation, each controller must contain it's own application program and/or a host PC must communicate with each controller individually. The number of controllers (and axes) that can be linked together via Ethernet is only limited by the number of IP addressed available. Controllers communicate with each other via the SA command. Groups of axes that are "tightly coupled" via vector mode (VM), linear interpolation (LM), multi-axis gearing (GA), or multi-axis camming (EA) must be grouped onto multi-axis controllers.	IH, SA

Each CDS-3310 provides two communication channels: RS-232 (up to 19.2K Baud) and 10/100BaseT Ethernet. The serial port may be replaced with a USB port as a factory option.

A 4Meg Flash EEPROM provides non-volatile memory for storing application programs, parameters, arrays and firmware. New firmware revisions are easily upgraded in the field.

Designed to solve complex motion problems, the CDS-3310 can be used for applications involving jogging, point-to-point positioning, electronic gearing to an encoder, multiple move sequences, and contouring. The controller eliminates jerk by programmable acceleration and deceleration with profile

smoothing. For smooth following of complex contours, the CDS-3310 provides continuous feed of an infinite number of linear segments (contour mode).

For synchronization with outside events, the CDS-3310 provides uncommitted I/O, including 8 digital inputs, 10 digital outputs, 2 analog intputs (0-5V) and one analog output (-10V to 10V). When the DB-28040 is added to the controller, the CDS-3310 has 8 analog inputs and an additional 40 digital I/O. Dedicated TTL inputs are provided for forward and reverse limit switches, a home switch, and abort.

Commands can be sent in either Binary or ASCII. Additional software is available for automatic-tuning, trajectory viewing on a PC screen, and program development using many environments such as Visual Basic, C, C++ etc. Drivers for DOS, Linux, QNX, Windows 3.1, 95, 98, 2000, ME, NT and XP are available.

Amplifier Specifications

The CDS-3310 integrated amplifier is a brush/brushless trans-conductance PWM amplifier. The amplifier operates in torque mode, and will output a motor current proportional to the command signal input from the controller. For high current applications (near 7 amps continuous), the metal enclosure should be mounted to metal backing to dissipate heat

***Note**: *Do not "hot swap" the motor power connections. If the amp is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector.

Supply Voltage:	18-72 VDC
Continuous Current:	7 Amps
Peak Current	10 Amps
Amplifier Gain	0.4, 0.7, or 1.0 A/V (selectable via AG command)
Switching Frequency	60 kHz
Minimum Load Inductance:	0.5 mH
Brushless Motor Commutation angle	120°

CDS-3310 Functional Elements

The CDS-3310 controller circuitry can be divided into the following functional groups as shown in Figure 1.1 and discussed below.

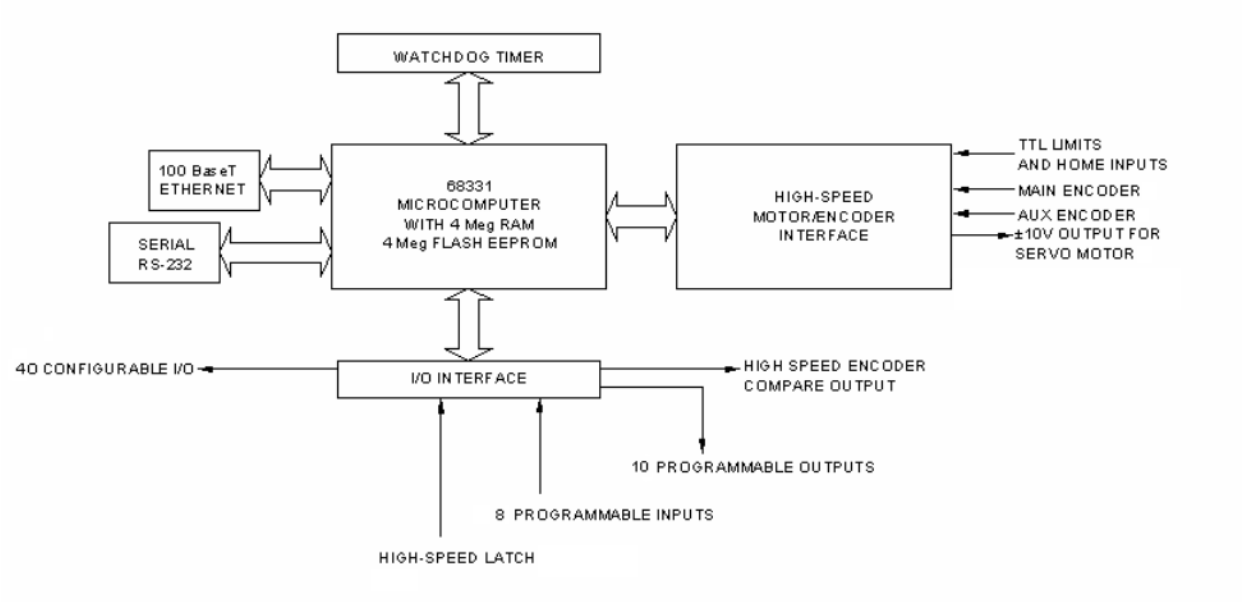


Figure 1.1 - CDS-3310 Functional Elements

Microcomputer Section

The main processing unit of the CDS-3310 is a specialized 32-Bit Motorola 68331 Series Microcomputer with 4 Meg RAM and 4 Meg Flash EEPROM. The RAM provides memory for variables, array elements and application programs. The flash EEPROM provides non-volatile storage of variables, programs, and arrays. It also contains the CDS-3310 firmware.

Motor Interface

Galil's GL-1800 custom, sub-micron gate array performs quadrature decoding of each encoder at up to 12 MHz. For standard servo operation, the controller generates a +/-10 Volt analog signal (16 Bit DAC).

Communication

The communication interface with the CDS-3310 consists of a RS-232 port and 10/100 BaseT Ethernet port. The RS-232 channel can generate up to 19.2Kbaud.

General I/O

The CDS-3310 provides interface circuitry for 8 TTL inputs and 10 TTL outputs. The CDS-3310 also has an additional 40 I/O daughterboard (DB-28040) that can be ordered as an option. General input 1 can also be used a the high speed position latch. A high speed encoder compare output is also provided.

System Elements

As shown in Fig. 1.2, the CDS-3310 controller and drive system is part of a motion control system, which includes motors and encoders. These elements are described below.

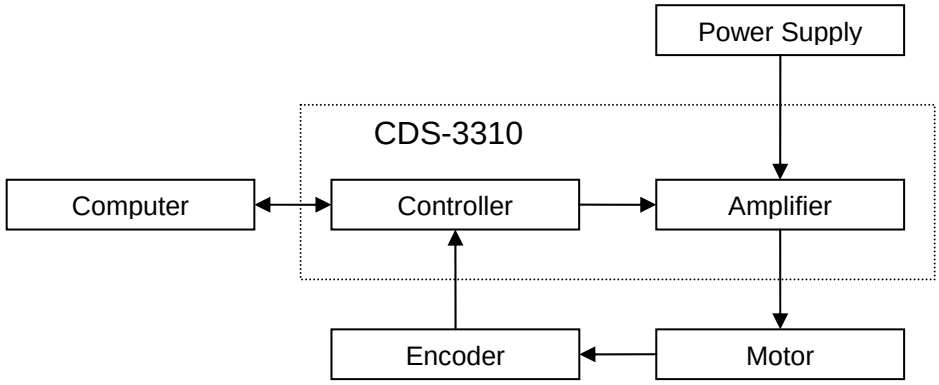


Figure 1.2 - Elements of Servo systems

Motor

A motor converts current into torque, which produces motion. Each axis of motion requires a motor sized properly to move the load at the required speed and acceleration. The Galil "Motion Component Selector" software can help with motor sizing. Contact Galil at 800-377-6329 if you would like this product.

The motor may be a servo motor and can be brush-type or brushless, rotary or linear.

Amplifier (Driver)

The power amplifier converts a +/-10 Volt signal from the controller into current to drive the motor. The amplifier should be sized properly to meet the power requirements of the motor. The amplifiers is pulse-width-modulated (PWM). The amplifier gain (AG) should be set such that a 10 Volt command generates the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V.

Encoder

An encoder translates motion into electrical pulses which are fed back into the controller. The CDS-3310 accepts feedback from either a rotary or linear encoder. Typical encoders provide two channels in quadrature, known as CHA and CHB. This type of encoder is known as a quadrature encoder. Quadrature encoders may be either single-ended (CHA and CHB) or differential (CHA,CHA- and CHB,CHB-). The CDS-3310 decodes either type into quadrature states or four times the number of cycles. Encoders may also have a third channel (or index) for synchronization.

There is no limit on encoder line density, however, the input frequency to the controller must not exceed 3,000,000 full encoder cycles/second (12,000,000 quadrature counts/sec). For example, if the encoder line density is 10000 cycles per inch, the maximum speed is 300 inches/second. If higher encoder frequency is required, please consult the factory.

The standard voltage level is TTL (zero to five volts), however, voltage levels up to 12 Volts are acceptable. (If using differential signals, 12 Volts can be input directly to the CDS-3310. Single-ended 12 Volt signals require a bias voltage input to the complementary inputs).

To interface with other types of position sensors such as resolvers or absolute encoders, Galil can customize the controller and command set. Please contact Galil and talk to one of our applications engineers about your particular system requirements.

Watch Dog Timer

The CDS-3310 provides an internal watch dog timer which checks for proper microprocessor operation. The timer toggles the Amplifier Enable Output (AMPEN) which can be used to switch the amplifier off in the event of a serious CDS-3310 failure. The AMPEN output is normally high. During power-up and if the microprocessor ceases to function properly, the AMPEN output will go low. The error light will also turn on at this stage. A reset is required to restore the CDS-3310 to normal operation. Consult the factory for a Return Materials Authorization (RMA) Number if your CDS-3310 is damaged.

Chapter 2 Getting Started

This section describes how to begin communicating with a single controller and how to begin controlling a single motor.

CDS-3310 Layout and Dimensions

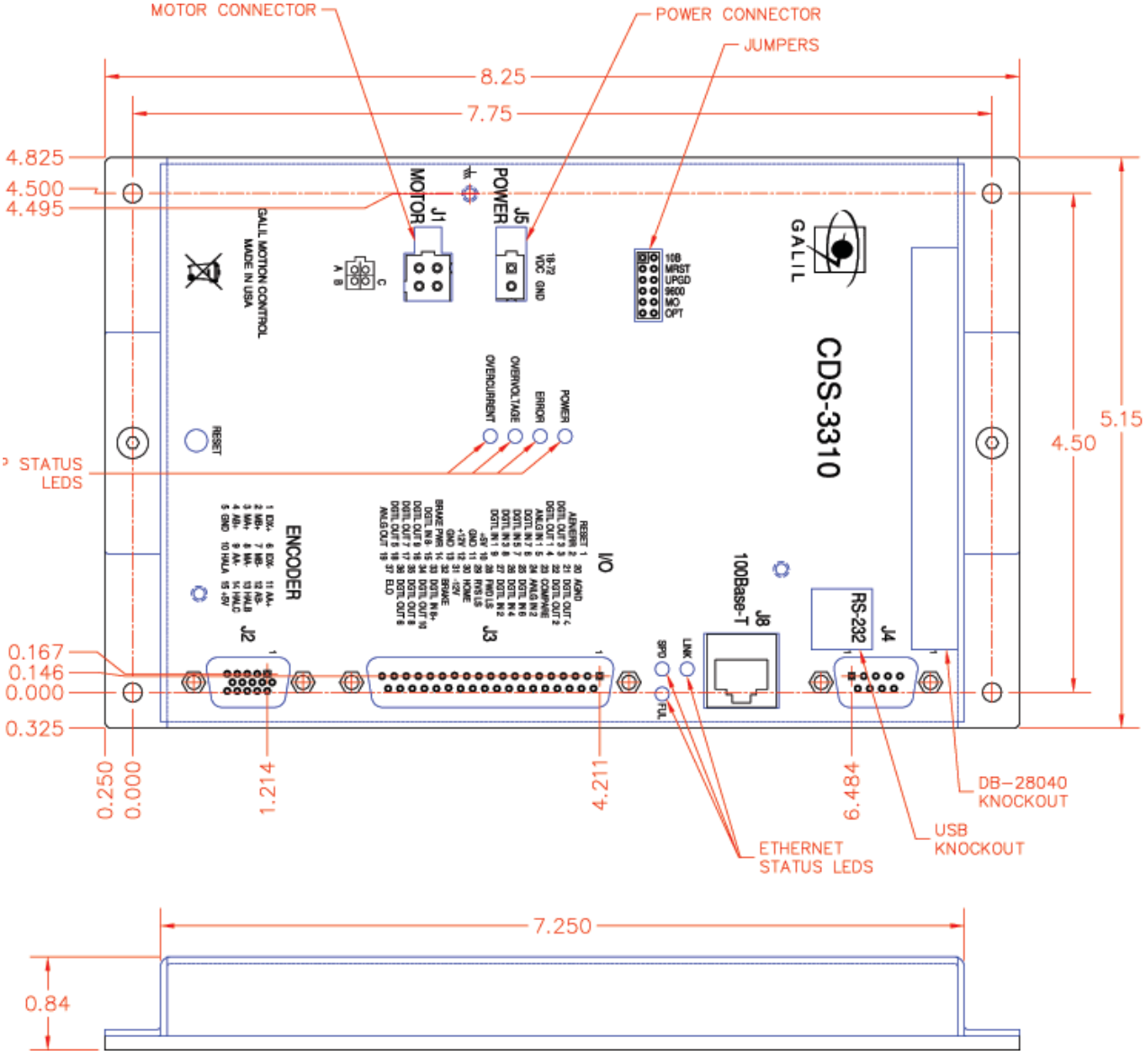


Figure 2-6 - CDS-3310

Recommended Components

For a complete system, Galil recommends the following elements:

1. CDS-3310 Motion Controller
2. ICM-3300 interconnect module with screw terminals and opto-isolation
3. Power Supply
4. Brush or Brushless Servo motor with Incremental Encoder

5. PC (Personal Computer - RS232 or Ethernet for CDS-3310)

6a. WSDK-32 (recommend for first time users.)

or

6b. Galil SmartTerminal communication software.

Software is available for download at http://www.galilmc.com/support/download.html.

Installing the CDS-3310

Step 1. Install jumpers on the CDS-3310

Step 2. Connect 18-72 VDC power to controller

Step 3. Install Windows communication software

Step 4. Establish communication

Step 5. Make connections to encoder

Step 6. Configure Amplifier

Step 7a. Connect brush servo motor

or

Step 7b. Connect brushless servo motor

Step 8. Close the Loop

Step 9. Tune the servo system

Step 1. Install Jumpers on the CDS-3310

Ethernet Speed (10B)

Install this jumper to force the Ethernet communication rate to 10 megabits per second. If the jumper is removed, the controller will auto negotiate either 10 or 100 megabits per second. This feature is available on CDS-3310 revision B or higher.

Master Reset (MRST)

When MRST is connected, the controller will perform a master reset upon power up or upon the reset input going low. Whenever the controller has a master reset, all programs, arrays, variables, and motion control parameters stored in EEPROM will be ERASED.

Upgrade Jumper (UPGD)

The UPGRD jumper enables the user to unconditionally update the controller's firmware. This jumper is not necessary for firmware updates when the controller is operating normally, but may be necessary in cases of corrupted EEPROM (e.g. power failure during a firmware update). If EEPROM corruption occurs, your controller may not operate properly. In this case, install the UPGRD Jumper and use the update firmware function on the Galil Terminal to re-load the system firmware.

Serial Port Baud Rate (9600)

The following table describes the baud rate settings:

9600	BAUD RATE
ON	9600
OFF	19200

Motor Off Jumper (MO)

The state of the motor upon power up may be selected with the placement of a hardware jumper on the controller. With a jumper installed at the MO location, the controller will be powered up in the "motor off" state. The SH command will need to be issued in order for the motor to be enabled. With no jumper installed, the controller will immediately enable the motor upon power up. The MO command will need to be issued to turn the motor off.

Option (OPT)

Reserved

Step 2. Connect 18 to 72 VDC Power to the Controller

Connect power to the connector labeled J5 POWER. You will need to crimp two pins onto the appropriate gage wire and insert the pins into an AMP mate-n-lock II connector.

Location	Mating Connector	Terminal Pins
J5 POWER (2-pin)	AMP# 172165-1	AMP# 170361-1

Connect the positive lead of your DC power supply to the pin labeled VDC on J5 and the negative lead to GND. Serious damage will occur if the power supply is incorrectly wired. The green power light indicator should go on when power is applied. Disconnect power when making wiring changes.

WARNING: Dangerous voltages, current, temperatures and energy levels exist in this product and the associated amplifiers and servo motor(s). Extreme caution should be exercised in the application of this equipment. Only qualified individuals should attempt to install, set up and operate this equipment.

Step 3. Install Windows Communication Software

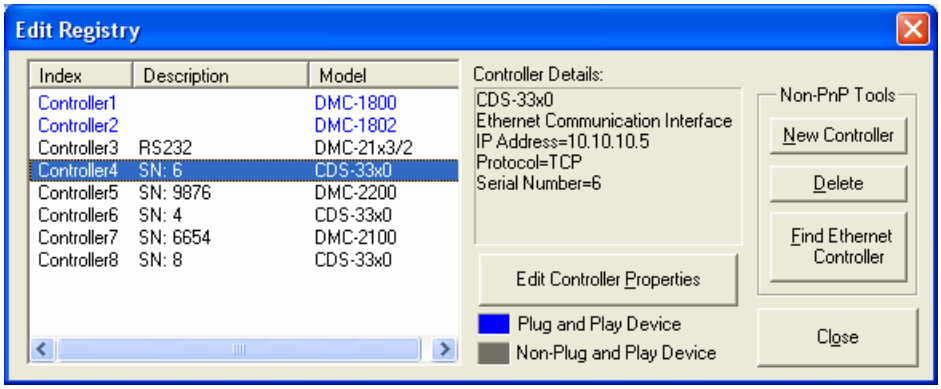
After applying power to the computer, you should install the Galil software that enables communication between the controller and PC. The following instructions apply to Windows 98 second edition, NT, ME, 2000 or XP. To install the basic communications software, run the Galil

Software CD-ROM and choose “DMCSmartTerm”. This will install the Galil Smart Terminal, which can be used for communication and programming of the controller.

Step 4. Establish Communication

In order for the Windows software to communicate with a Galil controller, the controller must be registered in the Windows Registry. The registry is accessed through the Galil software, such as WSDK or Galil Smart Terminal.

The registry window is equipped with buttons to Add a New Controller, change the Properties of an existing controller, Delete a controller, or Find an Ethernet Controller.



Step A1. Register a Serial Controller

Connect the CDS-3310 serial port to your computer via the Galil CABLE-9PIN-D (RS-232 “Straight Through” Serial Cable-- NOT Null Modem). The serial port is configured for full duplex, no parity, 8 data bits, one start bit, one stop bit, and hardware handshaking. See appendix for pin-out of serial cable

Use the New Controller button to add a new entry to the Registry. You will need to supply the Galil Controller model (e.g. CDS-3310). You then need to choose serial or Ethernet connection. The registry information will show a default Comm Port of 1 and a default Comm Speed of 19200 appears. This information can be changed as necessary to reflect the computer Comm Port and the baud rate set by the controller's IDC jumper (default is 19200).

Step A2. Register an Ethernet Controller

Use the Find Ethernet Controller button to have the software search for controllers connected to the network. If an IP address has not been assigned to the controller, click on Set IP Address and follow the directions on the screen. Refer to application note 4422 if you are having trouble.

Step B. Send Test Commands to the Terminal

Once the controller has been registered, select the correct controller from the list and click on OK. If the software successfully established communications with the controller, the registry entry will be displayed at the bottom of the screen in the Status window.

After you connect your terminal, press <return> or the <enter> key on your keyboard. In response to carriage return <return>, the controller responds with a colon. :

Now type

TPA <return>

This command directs the controller to return the current position of the A axis. The controller should respond with a number.

Step 5. Make Connections to Encoder

Step A. Connect Encoder Wires to 15-pin high density J2 ENCODER

The CDS-3310 accepts single-ended or differential encoder feedback with or without an index pulse. Match the leads from the encoder you are using to the encoder feedback inputs. The signal leads are labeled MA+ (channel A), MB+ (channel B), and IDX+ (index). For differential encoders, the complement signals are labeled MA-, MB-, and IDX-.

NOTE: When using pulse and direction encoders, the pulse signal is connected to channel A and the direction signal is connected to channel B. The controller must be configured for pulse and direction with the command CE (see command reference).

Step B. Verify proper encoder operation.

Once it is connected, turn the motor shaft and interrogate the position with the instruction TPA <return>. The controller response will vary as the motor is turned.

At this point, if TPA does not vary with encoder rotation, there are three possibilities:

1. The encoder connections are incorrect - check the wiring as necessary.
2. The encoder has failed - using an oscilloscope, observe the encoder signals. Verify that both channels A and B have a peak magnitude between 5 and 12 volts. Note that if only one encoder channel fails, the position reporting varies by one count only. If the encoder failed, replace the encoder. If you cannot observe the encoder signals, try a different encoder.
3. There is a hardware failure in the controller - connect the same encoder to a different controller. If the problem disappears, you probably have a hardware failure. Consult the factory for help.

Step 6. Configure Amplifier

Step A. Set the amplifier Gain

Select the amplifier gain that is appropriate for the motor. The amplifier gain command AG can be set to 0, 1, or 2 corresponding to 0.4, 0.7, and 1.0 A/V.

Step B. Set the Error Limit

Usually, there is uncertainty about the correct polarity of the feedback. The wrong polarity causes the motor to run away from the starting position. Using a terminal program, such as Galil SmartTerminal, the following parameters can be given to avoid system damage:

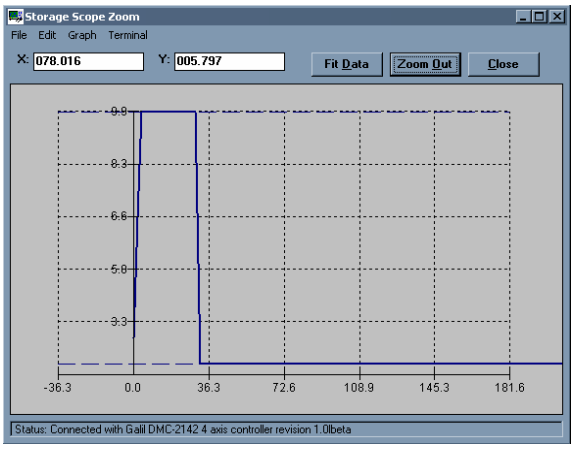
ER 2000 <CR> Sets error limit on the A axis to be 2000 encoder counts

OE 1 <CR> Disables A axis amplifier when excess position error exists

If the motor runs away and creates a position error of 2000 counts, the motor amplifier will be disabled.

Step C. Set Torque Limit

Peak and continuous torque limits can be set through TK and TL respectively. The TK and TL values are entered in volts on an axis by axis basis. The peak limit will set the maximum voltage that will be output from the controller to the amplifier. The continuous current will set what the maximum average current is over a short time interval. The following figure captured with WSDK is indicative of the continuous and peak operation. In this figure, the continuous limit was configured for 2 volts, and the peak limit was configured for 10 volts.



Peak Current Operation

Step 7a. Connect Brush Servo Motor

The CDS-3310 allows for brush operation. To configure an axis for brush-type operation, disconnect power and connect the two motor leads to Phase A and Phase B connections on J1 MOTOR and leave Phase C disconnected.

Location	Mating Connector	Terminal Pins
J1 MOTOR (4-pin)	AMP# 172167-1	AMP# 170361-1

Set the controller into brush-axis operation by issuing BR n,n,n,n. By setting n=1, the controller will operate in brushed mode on that axis. For example, BR0,1,0,0 sets the Y-axis as brush-type, all others as brushless. If an axis is set to brush-type, the amplifier has no need for the Hall inputs. These inputs can subsequently be used as general-use inputs and queried with the QH command.

Step 7b. Connect Brushless Servo Motor

Step A. Connect Hall Sensors

To configure an axis for brushless motor operation, connect the Hall sensors to pins HALA, HALB, and HALC on 15-pin high-density connector J2 ENCODER. The CDS-3310 can only be connected to a brushless motor with 120 degree Hall commutation. Verify that the hall sensors change state with the QH command.

Step B. Connect Motor Leads

Next, disconnect power and connect the three motor power leads to the phase A, B, and C connections on the J1 MOTOR connector.

Location	Mating Connector	Terminal Pins
J1 MOTOR (4-pin)	AMP# 172167-1	AMP# 170361-1

Step C1. Test wiring manually

The motor wiring must correspond to the Hall wiring, so if the motor manufacturer indicates how to do this, simply connect the power leads as specified. Otherwise, you will need to try up to six motor power lead combinations (keeping the hall sensor wiring fixed) until smooth motion results. Three of the combinations result in a stall condition (no torque), two result in

jerky motion in one direction but smooth in the other, and the last (correct) combination results in smooth movement in both directions. The following table will help you connect all six combinations:

Aid to connect all six motor power lead combinations

Phase A	Phase B	Phase C	Behavior
Power Lead 1	Power Lead 2	Power Lead 3	
Power Lead 1	Power Lead 3	Power Lead 2	
Power Lead 2	Power Lead 3	Power Lead 1	
Power Lead 2	Power Lead 1	Power Lead 3	
Power Lead 3	Power Lead 1	Power Lead 2	
Power Lead 3	Power Lead 2	Power Lead 1	

Set the PID gains to zero KPX=0; KDX=0; KIX=0. For each combination, set an offset OF2. If the motor does not move smoothly, go to the next combination. If it does move smoothly, try the other direction with OF-2. If it is smooth, you have found the correct combination. If not, go to the next combination.

Step C2. Test wiring automatically

The controller has been programmed to test whether the Hall commutation order is correct. To test the commutation for the X axis, issue the BS command (BSX= n,m). The controller will attempt to move the motor through one revolution. If the motor is unable to move, the controller will return "unknown Hall transition", check wiring, and execute BS again. It may be necessary to issue more voltage to create motion. The default for the BS command is BSn=0.25,1000 which will send 0.25 volts to the amplifier for 1 second. BSX=0.5,300 will issue 0.5 volts from the controller for 300 milliseconds. If the controller is able to move the motor and the Hall transitions are not correct, the controller will alert the operator and recommend which motor phases to change. For example, the controller might return "Wire A to Terminal B, Wire B to Terminal A." If the controller finds that the commutation order is correct, but the motor would run away due to positive feedback, the controller will prompt the user to "Wire Phase B to C and C to B. Exchange Hall Sensors A and B…". After making any necessary changes to the motor phase wiring, confirm correct operation by reissuing the BS command. Once the axis is wired correctly, the controller is ready to perform closed-loop motion.

Step 8. Close the Loop

Step A. Test feedback polarity

To test the polarity of the feedback, set small gains KPX=6; KDX=64 and command a move with the instruction:

PRX=1000 <CR> Position relative 1000 counts

BGX <CR> Begin motion on A axis

When the polarity of the feedback is wrong, the motor will run away and the controller will disable the motor when the position error exceeds 2000 counts. If the motor runs away, the polarity of the loop must be inverted.

Step B. Invert feedback polarity (if necessary)

When the polarity of the feedback is incorrect, the user must invert the loop polarity. The polarity reversal may be done with the encoder. If you are using a single-ended encoder, interchange the signal CHA and CHB. If, on the other hand, you are using a differential encoder, interchange only CHA+ and CHA-. The loop polarity and encoder polarity can also be affected through software with the MT, and CE commands (see the Command Reference).

The motor should now hold position.

Step C. Reverse positive direction (if desired)

Once the feedback polarity is correct (the motor does not run away), the positive direction of motion may be backwards from the desired positive direction. If this is the case, the positive direction can be changed with both the CE and MT commands while the motor is off. For example MOX; CEX=2; MTX=-1; SHX.

Once you have closed the loop with the correct polarity, you can move on to the compensation phase (servo system tuning) to adjust the PID filter parameters, KP, KD and KI. It is necessary to accurately tune your servo system to ensure fidelity of position and minimize motion oscillation as described in the next section.

Step 9. Tune the Servo System

Adjusting the tuning parameters is required when using servo motors. The system compensation provides fast and accurate response and the following presentation suggests a simple and easy way for compensation. More advanced design methods are available with software design tools from Galil, such as the Windows Servo Design Kit (WSDK software)

The filter has three parameters: the damping, KD; the proportional gain, KP; and the integrator, KI. The parameters should be selected in this order.

To start, set the gains to zero with the instructions

KIX= 0 <return> Integrator gain

KPX=0 <return> Proportional gain

KDX=0 <return> Derivative gain

Increase KD gradually (maximum is 4095) and stop after the motor vibrates. A vibration is noticed by audible sound or by interrogation. If you send the command

TEA <return> Tell error on A axis

a few times, and get varying responses, especially with reversing polarity, it indicates system vibration. When this happens, simply reduce KD.

Next you need to increase the value of KP gradually (maximum allowed is 1023). You can monitor the improvement in the response with the Tell Error instruction

KP 10 <return> Proportion gain

TEA <return> Tell error on A axis

As the proportional gain is increased, the error decreases.

Again, the system may vibrate if the gain is too high. In this case, reduce KP. Typically, KP should not be greater than KD/4.

Finally, to select KI, start with zero value and increase it gradually. The integrator eliminates the position error, resulting in improved accuracy. Therefore, the response to the instruction

TEA <return>

becomes zero. As KI is increased, its effect is amplified and it may lead to vibrations. If this occurs, simply reduce KI.

After appropriate PID gains are found, burn them into the EEPROM with BN.

For a more detailed description of the operation of the PID filter and/or servo system theory, see Chapter 10 - Theory of Operation.

Tuning must be done before configuring a distributed network (Chapter 11).

Design Examples

Here are a few examples for tuning and using your controller. These examples have remarks next to each command - these remarks must not be included in the actual program. Burn programs to EEPROM with the BP command.

System Set-up

This example assigns the proportional gain, error limit and enables the automatic error shut-off.

Instruction	Interpretation
KP10	Method for setting only A axis gain
OE 1	Enable automatic Off on Error function
ER1000	Set error limit to 1000 counts

Profiled Move

Rotate the axis a distance of 10,000 counts at a slew speed of 20,000 counts/sec and an acceleration and deceleration rates of 100,000 counts/s2. In this example, the motor turns and stops:

Instruction	Interpretation
PR1000	Distance
SP20000	Speed
DC 100000	Deceleration
AC 100000	Acceleration
BG	Start Motion

Position Interrogation

The position of the axis may be interrogated with the instruction, TP.

Instruction	Interpretation
TP	Tell position

The position error, which is the difference between the commanded position and the actual position can be interrogated with the instruction TE.

Instruction	Interpretation
TE	Tell error

Absolute Position

Objective: Command motion by specifying the absolute position.

Instruction	Interpretation
DP 2000	Define the current positions as 2000
PA 7000	Sets the desired absolute position
BG	Start A motion

After motion is complete, the axis can be command back to zero:

PA 0	Move to 0
BG	Start motion

Velocity Control

Objective: Drive the motor at specified speeds.

Instruction	Interpretation
JG20000	Set Jog Speed and Direction
AC 100000	Set acceleration
DC 50000	Set deceleration
BG	Start motion

after a few seconds, command:

JG -40000	New A speed and Direction
TV	Returns A speed

These cause velocity changes including direction reversal. The motion can be stopped with the instruction

ST	Stop

Operation Under Torque Limit

The magnitude of the motor command may be limited independently by the instruction TL.

Instruction	Interpretation
TL 0.2	Set output limit of A axis to 0.2 volts
JG 10000	Set A speed
BG	Start A motion

In this example, the A motor will probably not move since the output signal will not be sufficient to overcome the friction. If the motion starts, it can be stopped easily by a touch of a finger.

Increase the torque level gradually by instructions such as

Instruction	Interpretation
TL 1.0	Increase torque limit to 1 volt.
TL 9.998	Increase torque limit to maximum, 9.998 Volts.

The maximum level of 9.998 volts provides the full output torque.

Interrogation

The values of the parameters may be interrogated. Some examples …

Instruction	Interpretation
KP?	Return gain of A axis

Many other parameters such as KI, KD, FA, can also be interrogated. The command reference denotes all commands which can be interrogated.

Motion Programs with Loops

Motion programs may include conditional jumps as shown below.

Instruction	Interpretation
#A	Label
DP 0	Define current position as zero
v1=1000	Set initial value of v1

#LOOP	Label for loop
PA v1	Move A motor v1 counts
BG	Start A motion
AM	After A motion is complete
WT 500	Wait 500 ms
TP	Tell position A
v1=v1+1000	Increase the value of v1
JP #LOOP,v1<10001	Repeat if v1<10001
EN	End

After the above program is entered, download to the controller. To start the motion, command:

XQ #A	Execute Program #A

Control Variables

Objective: To show how control variables may be utilized.

Instruction	Interpretation
#A;DP0	Label; Define current position as zero
PR 4000	Initial position
SP 2000	Set speed
BG	Move A
AM	Wait until move is complete
WT 500	Wait 500 ms
#B	
v1 = _TPA	Determine distance to zero
PR -v1/2	Command A move 1/2 the distance
BGA	Start A motion
AMA	After A moved
WT 500	Wait 500 ms
v1=	Report the value of V1
JP #C, v1=0	Exit if position=0
JP #B	Repeat otherwise
#C	Label #C
EN	End of Program

To start the program, command

XQ #A	Execute Program #A

This program moves A to an initial position of 1000 and returns it to zero on increments of half the distance. Note, _TPA is an internal variable which returns the value of the A position. Internal variables may be created by preceding a CDS-3310 instruction with an underscore, _.

THIS PAGE LEFT PLANK INTENTIONALLY

Chapter 3 Connecting I/O

Overview

This chapter describes the inputs and outputs and their proper connection (see appendix if you are using an ICM-3300). In addition to encoder and hall inputs, the CDS-3310 provides the I/O listed below:

INPUTS

quantity	description
8	general
1	reverse limit
1	forward limit
1	home
1	abort (ELO)
1	reset
2	aux encoder
2	analog

OUTPUTS

quantity	description
10	general
1	amp enable/error
1	output compare
1	brake
1	analog

The DB-28040 can be used to add an additional 40 digital I/O and 8 analog inputs (see appendix).

Inputs

All digital inputs accept a 0 or 5V signal and are pulled up to 5V. To change the state of the input, the input must be shorted to ground (GND).

General Use Digital Inputs

The general use inputs are TTL and are labeled DGTL IN 1 to DGTL IN 8 on the silkscreen on the sheet metal. These inputs can be interrogated with the use of the command TI (Tell Inputs), the operand _TI, and the function @IN[n] (see Chapter 7, Mathematical Functions and Expressions).

Digital input 8 can accept a differential (two-wire) signal. To connect a single-ended (one wire) signal, connect to DGTL IN 8+ and leave DGTL IN 8- disconnected.

Limit Switch Inputs

The forward limit switch (FLSx) inhibits motion in the forward direction immediately upon activation of the switch. The reverse limit switch (RLSx) inhibits motion in the reverse direction immediately upon activation of the switch. If a limit switch is activated during motion, the controller will make a decelerated stop using the deceleration rate previously set with the DC command. The motor will remain on (in a servo state) after the limit switch has been activated and will hold motor position. To

set the activation state of the limit switches refer to the command CN, configure, in the Command Reference.

When a forward or reverse limit switch is activated, the current application program that is running will be interrupted and the controller will automatically jump to the #LIMSWI subroutine if one exists. This is a subroutine which the user can include in any motion control program and is useful for executing specific instructions upon activation of a limit switch.

After a limit switch has been activated, further motion in the direction of the limit switch will not be possible until the logic state of the switch returns back to an inactive state. This usually involves physically opening the tripped switch. Any attempt at further motion before the logic state has been reset will result in the following error: "022 - Begin not possible due to limit switch" error.

The operands, _LFx and _LRx, return the state of the forward and reverse limit switches, respectively (x represents the axis, X,Y,Z, or W). The value of the operand is either a '0' or '1' corresponding to the logic state of the limit switch, active or inactive, respectively. If the limit switches are configured for active low, no connection or a 5V input will be read as a '0', while grounding the switch will return a '1'. If the limit switches are configured for active high, the reading will be inverted and no connection or a 5V input will be read as a '1', while grounding the switch will return a '0'.

Using a terminal program, the state of a limit switch can be printed to the screen with the command, MG _LFx or MG _LRx. This prints the value of the limit switch operands for the 'x' axis. The logic state of the limit switches can also be interrogated with the TS command. For more details on TS, _LFx, _LRx, or MG see the Command Reference.

Home Switch Input

Homing inputs are designed to provide mechanical reference points for a motion control application. A transition in the state of a Home input alerts the controller that a particular reference point has been reached by a moving part in the motion control system. A reference point can be a point in space or an encoder index pulse.

The Home input detects any transition in the state of the switch and changes between logic states 0 and 1, corresponding to either 0V or 5V depending on the configuration set by the user (CN command). The CN command can be used to customize the homing routine to the user's application.

There are three homing routines supported by the CDS-3310: Find Edge (FE), Find Index (FI), and Standard Home (HM).

The Find Edge routine is initiated by the command sequence: FEX <return>, BGX <return> (where X could be any axis on the controller, X,Y,Z, or W). The Find Edge routine will cause the motor to accelerate, then slew at constant speed until a transition is detected in the logic state of the Home input. The direction of the FE motion is dependent on the state of the home switch. Refer to the CN command to set the correspondence between the Home Input voltage and motion direction. The motor will decelerate to a stop when a transition is seen on the input. The acceleration rate, deceleration rate and slew speed are specified by the user, prior to the movement, using the commands AC, DC, and SP. *It is recommended that a high deceleration value be used so the motor will decelerate rapidly after sensing the Home switch.*

The Find Index routine is initiated by the command sequence: FIX <return>, BGX <return> (where X could be any axis on the controller, X,Y,Z, or W). Find Index will cause the motor to accelerate to the user-defined slew speed (SP) at a rate specified by the user with the AC command and slew until the controller senses a change in the index pulse signal from low to high. The motor then decelerates to a stop at the rate previously specified by the user with the DC command. *Although Find Index is an option for homing, it is not dependent upon a transition in the logic state of the Home input, but instead is dependent upon a transition in the level of the index pulse signal.*

The Standard Homing routine is initiated by the sequence of commands HMX <return>, BGX <return> (where X could be any axis on the controller, X,Y,Z, or W). Standard Homing is a combination of Find Edge and Find Index homing. Initiating the standard homing routine will cause

the motor to slew until a transition is detected in the logic state of the Home input. The motor will accelerate at the rate specified by the command, AC, up to the slew speed. After detecting the transition in the logic state on the Home Input, the motor will decelerate to a stop at the rate specified by the command DC. After the motor has decelerated to a stop, it switches direction and approaches the transition point at the speed of 256 counts/sec. When the logic state changes again, the motor moves forward (in the direction of increasing encoder count) at the same speed, until the controller senses the index pulse. After detection, it decelerates to a stop and defines this position as 0. The logic state of the Home input can be interrogated with the command MG _HMX. This command returns a 0 or 1 if the logic state is low or high (dependent on the CN command). The state of the Home input can also be interrogated indirectly with the TS command.

For examples and further information about Homing, see command HM, FI, FE of the Command Reference and the section entitled 'Homing' in the Programming Motion Chapter of this manual.

Abort Input

The function of the Abort input is to immediately stop the controller upon transition of the logic state.

NOTE: The response of the abort input is significantly different from the response of an activated limit switch. When the abort input is activated, the controller stops generating motion commands immediately, whereas the limit switch response causes the controller to make a decelerated stop.

NOTE: The effect of an Abort input is dependent on the state of the off on error function (OE) for each axis. If the Off On Error function is enabled for any given axis, the motor for that axis will be turned off when the abort signal is generated. This could cause the motor to 'coast' to a stop since it is no longer under servo control. If the Off On Error function is disabled, the motor will decelerate to a stop as fast as mechanically possible and the motor will remain in a servo state.

All motion programs that are currently running are terminated when a transition in the Abort input is detected. For information on setting the Off On Error function, see the Command Reference, OE.

Reset Input

Bringing the reset inputs low is equivalent to power cycling the controller.

Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. The controller has one auxiliary encoder and each auxiliary encoder consists of two inputs, channel A and channel B. The auxiliary encoder inputs are mapped to the inputs 81-92.

Each input from the auxiliary encoder is a differential line receiver and can accept voltage levels between +/- 12Volts. The inputs have been configured to accept TTL level signals. To connect TTL signals, simply connect the signal to the + input and leave the - input disconnected. For other signal levels, the - input should be connected to a voltage that is ½ of the full voltage range (for example, connect the - input to 6 volts if the signal is a 0 - 12 volt logic).

Example:

A CDS-3310 has one auxiliary encoder. This encoder has two inputs (channel A and channel B). Channel A input is mapped to input 81 and Channel B input is mapped to input 82. To use this input for 2 TTL signals, the first signal will be connected to AA+ and the second to AB+. AA- and AB- will be left unconnected. To access this input, use the function @IN[81] and @IN[82].

Analog Inputs

The CDS-3310 has two analog inputs configured for the range between 0V and 5V. The inputs are decoded by a 12-bit A/D decoder giving a voltage resolution of approximately 1 mV (a 16-bit A/D is available on the DB-28040). The impedance of these inputs is effectively infinite. The analog inputs are read with @AN[x] where x is a number 1 thru 2.

Outputs

The general outputs, amp enable, output compare, and error outputs are 0 or 5V digital outputs:

General Use Digital Outputs

The general use outputs are TTL and are labeled DGTL OUT 1 to DGTL OUT 10 on the silkscreen on the sheet metal. These outputs can be turned On and Off with the commands, SB (Set Bit), CB (Clear Bit), OB (Output Bit), and OP (Output Port). The value of the outputs can be checked with the operand _OP and the function @OUT[] (see Chapter 7, Mathematical Functions and Expressions).

Amp Enable

This signal is used by the controller internally to disable the amplifier and is exposed for optional external connection. The signal toggles with the commands MO and SH.

Output Compare

The output compare signal is TTL and is labeled COMPARE on the silkscreen. Output compare is controlled by the position of any of the main encoders on the controller. The output can be programmed to produce an active low pulse (1usec) based on an incremental encoder value or to activate once when an axis position has been passed. For further information, see the command OC in the Command Reference.

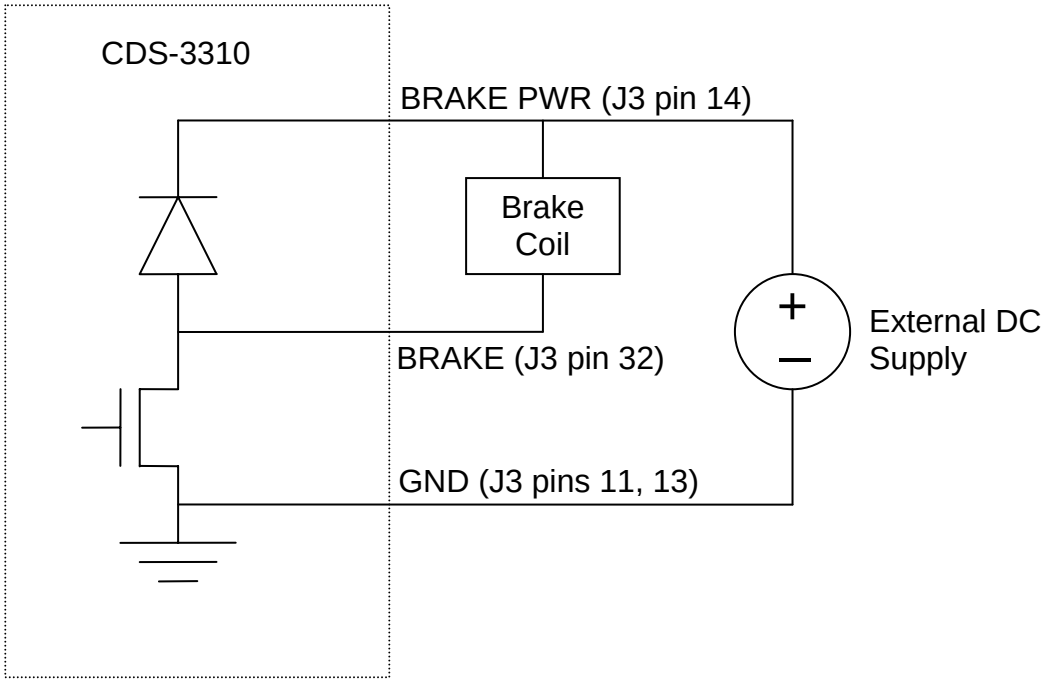
Error Output

The error output is TTL and is available on the breakout modules to indicate a controller error condition. When an error condition occurs, the ERROR signal will go low and the controller LED will go on. An error occurs because of one of the following conditions:

1. The axis has a position error greater than the error limit. The error limit is set by using the command ER.
2. The reset line on the controller is held low or is being affected by noise.
3. There is a failure on the controller and the processor is resetting itself.
4. There is a failure with the output IC which drives the error signal.

Brake Output

The brake digital output is provided for direct connection to a normally-engaged motor brake and external DC power supply (5 to 24 V). The brake is actuated by a MOSFET that can sink up to 500 mA. The controller provides an internal snubbing diode via the brake power input. The diagram below shows how to wire a brake. The brake is controlled with the MO (FET off) and SH (FET on) commands. The OS command allows the brake to be controlled with SB and CB.



Brake Output Wiring Diagram

Analog Output

The CDS-3310 has one analog output configured for the range between -10V and 10V. The output is driven by a 16-bit D/A converter giving a voltage resolution of approximately 300 μV. The analog output is set with AO command.

THIS PAGE LEFT PLANK INTENTIONALLY

Chapter 4 Communication

Introduction

The CDS-3310 has one RS232 port and one Ethernet port. The RS-232 port can be configured to speeds of up to 19200 baud. The Ethernet port is 10/100baseT.

RS232 Port

Configure your PC for 8-bit data, one start-bit, one stop-bit, full duplex and no parity. The Baud rate is set by installing a jumper. The RS232 pin-out and jumper configuration is given in the appendix.

Ethernet Configuration

Communication Protocols

The Ethernet is a local area network through which information is transferred in units known as packets. Communication protocols are necessary to dictate how these packets are sent and received. The CDS-3310 supports two industry standard protocols, TCP/IP and UDP/IP. The controller will automatically respond in the format in which it is contacted.

TCP/IP is a "connection" protocol. The master must be connected to the slave in order to begin communicating. Each packet sent is acknowledged when received. If no acknowledgement is received, the information is assumed lost and is resent.

Unlike TCP/IP, UDP/IP does not require a "connection". This protocol is similar to communicating via RS232. If information is lost, the controller does not return a colon or question mark. Because the protocol does not provide for lost information, the sender must re-send the packet.

Although UDP/IP is more efficient and simple, Galil recommends using the TCP/IP protocol. TCP/IP insures that if a packet is lost or destroyed while in transit, it will be resent.

Ethernet communication transfers information in 'packets'. The packets must be limited to 470 data bytes or less. Larger packets could cause the controller to lose communication.

NOTE: In order not to lose information in transit, Galil recommends that the user wait for an acknowledgement of receipt of a packet before sending the next packet.

Addressing

There are three levels of addresses that define Ethernet devices. The first is the Ethernet or hardware MAC address. This is a unique and permanent 6 byte number. No other device will have the same

Ethernet address. The CDS-3310 Ethernet address is set by the factory and the last two bytes of the address are the serial number of the controller.

The second level of addressing is the IP address. This is a 32-bit (or 4 byte) number. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the controller can be done in a number of ways.

The first method is to use the BOOT-P utility via the Ethernet connection (the CDS-3310 must be connected to network and powered). For a brief explanation of BOOT-P, see the section: *Third Party Software*. Either a BOOT-P server on the internal network or the Galil terminal software may be used (see Chapter 2).

CAUTION: Be sure that there is only one BOOT-P server running. If your network has DHCP or BOOT-P running, it may automatically assign an IP address to the controller upon linking it to the network. In order to ensure that the IP address is correct, please contact your system administrator before connecting the controller to the Ethernet network.

The second method for setting an IP address is to send the IA command through the CDS-3310 RS-232 port. The IP address you want to assign may be entered as a 4 byte number delimited by commas (industry standard uses periods) or a signed 32 bit number. (Ex. IA 124,51,29,31 or IA 2083724575) Type in BN to save the IP address to the controller's non-volatile memory.

NOTE: Galil strongly recommends that the IP address selected is not one that can be accessed across the Gateway. The Gateway is an application that controls communication between an internal network and the outside world. To decide on an IP address, find the address of the computer the controller will be connecting to, then find the computer's Subnet Mask. (In Windows, open a DOS prompt and type ipconfig). The fields of the subnet mask that contain a "255" require that the corresponding fields in the IP address of the controller be the same as the IP address of the PC. For instance, if the Subnet Mask of the PC is 255.255.0.0, and the IP address is 192.168.100.10 – then the controller's IP address should be set to 192.168.nnn.nnn where nnn can be anything from 0-255.

The third level of Ethernet addressing is the UDP or TCP port number. The Galil controller does not require a specific port number. The port number is established by the client or master each time it connects to the controller.

Communicating with Multiple Devices (Handles)

The CDS-3310 is capable of supporting multiple masters and slaves. The masters may be multiple PC's that send commands to the controller. The slaves are typically peripheral I/O devices that receive commands from the controller.

NOTE: The term "Master" is equivalent to the internet "client". The term "Slave" is equivalent to the internet "server".

An Ethernet handle is a communication resource within a device. The CDS-3310 can have a maximum of 8 Ethernet handles open at any time. The TH command may be used to indicate which handles are currently connected to and which are currently free. When using TCP/IP, each master or slave uses an individual Ethernet handle. In UDP/IP, one handle may be used for all the masters, but each slave uses one. (Pings and ARP's do not occupy handles.) If all 8 handles are in use and a 9th master tries to connect, it will be sent a "reset packet" that generates the appropriate error in its windows application.

When the Galil controller acts as the master, the IH command is used to assign handles and connect to its slaves. The IP address may be entered as a 4 byte number separated with commas (industry standard uses periods) or as a signed 32 bit number. A port number may also be specified, but if it is not, it will default to 1000. The protocol (TCP/IP or UDP/IP) to use must also be designated at this time. Otherwise, the controller will not connect to the slave. (Ex. IHB=151,25,255,9<179>2 This will open handle B and connect to the IP address 151.25.255.9, port 179, using TCP/IP).

If the slave is another Galil Ethernet controller, the SA command can be used to send commands as in the following example:

```
IHC=10,10,10,20          ;'open a handle to another Galil controller
#L; JP#L, _IHC2<>-2      ;'wait until the connection is complete
SAC = "MG _TPX"          ;'sent a command to the controller
WT5                      ;'wait for the response
Position = _SAC0         ;'assign the result to a variable
```

Modbus

An additional protocol layer is available for speaking to I/O devices. Modbus over TCP is a protocol that packages information in binary packets that are sent as part of a TCP/IP packet. In this protocol, each slave has a 1 byte slave address. The CDS-3310 can use a specific slave address or default to the handle number. The port number for Modbus is 502. The Modbus protocol has a set of commands called function codes. The CDS-3310 supports the 10 major function codes:

Function Code	Definition
01	Read Coil Status (Read Bits)
02	Read Input Status (Read Bits)
03	Read Holding Registers (Read Words)
04	Read Input Registers (Read Words)
05	Force Single Coil (Write One Bit)
06	Preset Single Register (Write One Word)
07	Read Exception Status (Read Error Code)
15	Force Multiple Coils (Write Multiple Bits)
16	Preset Multiple Registers (Write Words)
17	Report Slave ID

The CDS-3310 provides three levels of Modbus communication. The first level allows the user to create a raw packet and receive raw data. It uses the MBh command with a function code of –1. The format of the command is

```
MBh = -1,len,array[]      where   len is the number of bytes
                                  array[] is the array with the data
```

The second level incorporates the Modbus structure. This is necessary for sending configuration and special commands to an I/O device. The formats vary depending on the function code that is called. For more information refer to the Command Reference.

The third level of Modbus communication uses standard Galil commands. Once the slave has been configured, the commands that may be used are @IN[], SB, CB, OB, and AO. For example, AO 2020,8.2 would tell I/O number 2020 to output 8.2 volts.

If a specific slave address is not necessary, the I/O number to be used can be calculated with the following:

```
I/O Number = (HandleNum*1000) +((Module-1)*4) + (BitNum-1)
```

Where HandleNum is the handle number from 1 (A) to 8 (H). Module is the position of the module in the rack from 1 to 16. BitNum is the I/O point in the module from 1 to 4.

If an explicit slave address is to be used, the equation becomes:

I/O Number = (SlaveAddress*10000) + (HandleNum*1000) +((Module-1)*4) + (Bitnum-1)

To view an example procedure for communicating with an OPTO-22 rack, refer to the appendix.

Which devices receive what information from the controller depends on a number of things. If a device queries the controller, it will receive the response unless it explicitly tells the controller to send it to another device. If the command that generates a response is part of a downloaded program, the response will route to whichever port is specified as the default (unless explicitly told to go to another port) with the ENET switch ("ON" designates Ethernet in which case it goes to the last handle to communicate with the controller, "OFF" designates main RS232). To designate a specific destination for the information, add {Eh} to the end of the command. (Ex. MG{EC}"Hello" will send the message "Hello" to handle #3. TP,,?{EF} will send the z axis position to handle #6.)

Multicasting

A multicast may only be used in UDP/IP and is similar to a broadcast (where everyone on the network gets the information) but specific to a group. In other words, all devices within a specified group will receive the information that is sent in a multicast. There can be many multicast groups on a network and are differentiated by their multicast IP address. To communicate with all the devices in a specific multicast group, the information can be sent to the multicast IP address rather than to each individual device IP address. All Galil controllers belong to a default multicast address of 239.255.19.56. The controller's multicast IP address can be changed by using the IA> u command.

Using Third Party Software

Galil supports ARP, BOOT-P, and Ping which are utilities for establishing Ethernet connections. ARP is an application that determines the Ethernet (hardware) address of a device at a specific IP address. BOOT-P is an application that determines which devices on the network do not have an IP address and assigns the IP address you have chosen to it. Ping is used to check the communication between the device at a specific IP address and the host computer.

The CDS-3310 can communicate with a host computer through any application that can send TCP/IP or UDP/IP packets. A good example of this is Telnet, a utility that comes with most Windows systems.

Data Record

The CDS-3310 can provide a block of status information with the use of a single command, QR. This command, along with the QZ command can be very useful for accessing complete controller status. The QR command will return 4 bytes of header information and specific blocks of information as specified by the command arguments:

QR ABCDEFGHST

Each argument corresponds to a block of information according to the Data Record Map below. If no argument is given, the entire data record map will be returned. Note that the data record size will depend on the number of axes. See dmcwin/include/dmcdrc.h.

Data Record Map

DATA TYPE	ITEM	BLOCK
UB	1st byte of header	Header
UB	2nd byte of header	Header
UB	3rd byte of header	Header
UB	4rth byte of header	Header
UW	sample number	I block
UB	general input 0	I block
UB	general input 1	I block
UB	general input 2	I block
UB	general input 3	I block
UB	general input 4	I block
UB	general input 5	I block
UB	general input 6	I block
UB	general input 7	I block
UB	general input 8	I block
UB	general input 9	I block
UB	general output 0	I block
UB	general output 1	I block
UB	general output 2	I block
UB	general output 3	I block
UB	general output 4	I block
UB	general output 5	I block
UB	general output 6	I block
UB	general output 7	I block
UB	general output 8	I block
UB	general output 9	I block
UB	error code	I block
UB	general status	I block
UW	reserved	S block
UW	reserved	S block
SL	reserved	S block
UW	reserved	T block

UW	reserved	T block
SL	reserved	T block
UW	a axis status	A block
UB	a axis switches	A block
UB	a axis stopcode	A block
SL	a axis reference position	A block
SL	a axis motor position	A block
SL	a axis position error	A block
SL	a axis auxiliary position	A block
SL	a axis velocity	A block
SW	a axis torque	A block
SW	0 or a axis analog (DB-28040)	A block
UW	b axis status	B block
UB	b axis switches	B block
UB	b axis stopcode	B block
SL	b axis reference position	B block
SL	b axis motor position	B block
SL	b axis position error	B block
SL	b axis auxiliary position	B block
SL	b axis velocity	B block
SW	b axis torque	B block
SW	0 or b axis analog (DB-28040)	B block
UW	c axis status	C block
UB	c axis switches	C block
UB	c axis stopcode	C block
SL	c axis reference position	C block
SL	c axis motor position	C block
SL	c axis position error	C block
SL	c axis auxiliary position	C block
SL	c axis velocity	C block
SW	c axis torque	C block
SW	0 or c axis analog (DB-28040)	C block
UW	d axis status	D block
UB	d axis switches	D block
UB	d axis stopcode	D block
SL	d axis reference position	D block
SL	d axis motor position	D block
SL	d axis position error	D block
SL	d axis auxiliary position	D block
SL	d axis velocity	D block
SW	d axis torque	D block
SW	0 or d axis analog (DB-28040)	D block
UW	e axis status	E block
UB	e axis switches	E block
UB	e axis stopcode	E block

SL	e axis reference position	E block
SL	e axis motor position	E block
SL	e axis position error	E block
SL	e axis auxiliary position	E block
SL	e axis velocity	E block
SW	e axis torque	E block
SW	0 or e axis analog (DB-28040)	E block
UW	f axis status	F block
UB	f axis switches	F block
UB	f axis stopcode	F block
SL	f axis reference position	F block
SL	f axis motor position	F block
SL	f axis position error	F block
SL	f axis auxiliary position	F block
SL	f axis velocity	F block
SW	f axis torque	F block
SW	0 or f axis analog (DB-28040)	F block
UW	g axis status	G block
UB	g axis switches	G block
UB	g axis stopcode	G block
SL	g axis reference position	G block
SL	g axis motor position	G block
SL	g axis position error	G block
SL	g axis auxiliary position	G block
SL	g axis velocity	G block
SW	g axis torque	G block
SW	0 or g axis analog (DB-28040)	G block
UW	h axis status	H block
UB	h axis switches	H block
UB	h axis stopcode	H block
SL	h axis reference position	H block
SL	h axis motor position	H block
SL	h axis position error	H block
SL	h axis auxiliary position	H block
SL	h axis velocity	H block
SW	h axis torque	H block
SW	0 or h axis analog (DB-28040)	H block

NOTE: UB = Unsigned Byte, UW = Unsigned Word, SW = Signed Word, SL = Signed Long Word

Explanation of Status and Axis Switch Information

Header Information – Byte 0, 1 of Header:

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
1	N/A	N/A	N/A	N/A	I Block Present	T Block Present	S Block Present
BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
H Block Present	G Block Present	F Block Present	E Block Present	D Block Present	C Block Present	B Block Present	A Block Present

Bytes 2, 3 of Header:

Bytes 2 and 3 make a word, which represents the Number of bytes in the data record, including the header.

Byte 2 is the low byte and byte 3 is the high byte

NOTE: The header information of the data records is formatted in little endian.

General Status Information (1 Byte)

BIT 7	BIT6	BIT5	BIT4	BIT3	BIT 2	BIT 1	BIT 0
Program Running	N/A	N/A	N/A	N/A	Waiting for input from IN command	Trace On	Echo On

Axis Switch Information (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Latch Occurred	State of Latch Input	N/A	N/A	State of Forward Limit	State of Reverse Limit	State of Home Input	N/A

Axis Status Information (2 Bytes)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	Mode of Motion PA or PR	Mode of Motion PA only	(FE) Find Edge in Progress	Home (HM) in Progress	1st Phase of HM complete	2nd Phase of HM complete or FI command issued	Mode of Motion Coord. Motion
BIT 7	**BIT 6**	**BIT 5**	**BIT 4**	**BIT 3**	**BIT 2**	**BIT 1**	**BIT 0**
Negative Direction Move	Mode of Motion Contour	Motion is slewing	Motion stopping due to ST or Limit Switch	Motion is making final decel.	Latch is armed	Off On Error armed	Motor Off

Notes Regarding Velocity and Torque Information

The velocity information that is returned in the data record is 64 times larger than the value returned when using the command TV (Tell Velocity). See command reference for more information about TV.

The Torque information is represented as a number in the range of +/-32767. Maximum negative torque is -32767. Maximum positive torque is 32767. Zero torque is 0.

QZ Command

The QZ command can be very useful when using the QR command, since it provides information about the controller and the data record. The QZ command returns the following 4 bytes of information.

BYTE #	INFORMATION
0	Number of axes present
1	number of bytes in general block of data record
2	number of bytes in coordinate plane block of data record
3	Number of Bytes in each axis block of data record

Controller Response to Commands

Most CDS-3310 instructions are represented by two characters followed by the appropriate parameters. Each instruction must be terminated by a carriage return or semicolon.

Instructions are sent in ASCII, and the CDS-3310 decodes each ASCII character (one byte) one at a time. It takes approximately 0.5 msec for the controller to decode each command. However, the PC can send data to the controller at a much faster rate because of the FIFO buffer.

After the instruction is decoded, the CDS-3310 returns a response to the port from which the command was generated. If the instruction was valid, the controller returns a colon (:) or a question mark (?) if the instruction was not valid. For example, the controller will respond to commands which are sent via the main RS-232 port back through the RS-232 port, and to commands which are sent via the Ethernet port back through the Ethernet port.

For instructions that return data, such as Tell Position (TP), the CDS-3310 will return the data followed by a carriage return, line feed and colon.

It is good practice to check for : after each command is sent to prevent errors. An echo function is provided (serial only) to enable associating the CDS-3310 response with the data sent. The echo is enabled by sending the command EO 1 to the controller.

Unsolicited Messages Generated by Controller

When the controller is executing a program, it may generate responses which will be sent via the main RS-232 port or Ethernet ports. This response could be generated as a result of messages using the MG or IN (serial only) command OR as a result of a command error. These responses are known as unsolicited messages since they are not generated as the direct response to a command.

Messages can be directed to a specific port using the CF command or using the specific Port arguments – see MG and IN commands described in the Command Reference. If the port is not explicitly given, unsolicited messages will be sent to the default port.

The controller has a special command, CW, which can affect the format of unsolicited messages. This command is used by Galil Software to differentiate responses from the command line and unsolicited messages. The command CW1 causes the controller to set the high bit of ASCII characters to 1 of all unsolicited characters. This may cause characters to appear garbled to some terminals. This function can be disabled by issuing CW2. For more information, see application note 2437.

Galil Software Tools and Libraries

API (Application Programming Interface) software is available from Galil. The API software is written in C and is included in the Galil CD-ROM. They can be used for development under DOS and Windows environments (16 and 32 bit Windows). With the API's, the user can incorporate already existing library functions directly into a C program.

Galil has also developed an ActiveX Tool Kit. This provides 32-bit OCXs for handling all of the CDS-3310 communications including support of interrupts. These objects install directly into Visual Basic, LabVIEW, C, or any software that accepts ActiveX tools and are part of the run-time environment. For more information, contact Galil.

Chapter 5 Command Basics

Introduction

The CDS-3310 provides over 100 commands for specifying motion and machine parameters. Commands are included to initiate action, interrogate status and configure the digital filter. These commands can be sent in ASCII or binary.

In ASCII, the CDS-3310 instruction set is BASIC-like and easy to use. Instructions consist of two uppercase letters that correspond phonetically with the appropriate function. For example, the instruction BG begins motion, and ST stops the motion. In binary, commands are represented by a binary code ranging from 80 to FF.

ASCII commands can be sent "live" over the bus for immediate execution by the CDS-3310, or an entire group of commands can be downloaded into the CDS-3310 memory for execution at a later time. Combining commands into groups for later execution is referred to as Applications Programming and is discussed in the following chapter. Binary commands cannot be used in Applications programming.

This section describes the CDS-3310 instruction set and syntax. A summary of commands as well as a complete listing of all CDS-3310 instructions is included in the *Command Reference Manual*.

Command Syntax - ASCII

CDS-3310 instructions are represented by two ASCII upper case characters followed by applicable arguments. A space may be inserted between the instruction and arguments. A semicolon or <return> is used to terminate the instruction for processing by the CDS-3310 command interpreter.

NOTE: If you are using a Galil terminal program, commands will not be processed until an <return> command is given. This allows the user to separate many commands on a single line and not begin execution until the user gives the <return> command.

IMPORTANT: All CDS-3310 commands are sent in upper case.

For example, the command

PR 4000 <return> Position Relative

PR is the two character instruction for position relative. 4000 is the argument which represents the required position value in counts. The <return> terminates the instruction. The space between PR and 4000 is optional.

For specifying data for the A,B,C and D axes, commas are used to separate the axes. If no data is specified for an axis, a comma is still needed as shown in the examples below. If no data is specified for an axis, the previous value is maintained.

To view the current values for each command, type the command followed by a ? for each axis requested.

PR 1000	Specify A only as 1000
PR ,2000	Specify B only as 2000
PR ,,3000	Specify C only as 3000
PR ,,,4000	Specify D only as 4000
PR 2000, 4000,6000, 8000	Specify A,B,C and D
PR ,8000,,9000	Specify B and D only
PR ?,?,?,?	Request A,B,C,D values
PR ,?	Request B value only

The CDS-3310 provides an alternative method for specifying data. Here data is specified individually using a single axis specifier such as A,B,C or D. An equals sign is used to assign data to that axis. For example:

PRA=1000	Specify a position relative movement for the A axis of 1000
ACB=200000	Specify acceleration for the B axis as 200000

Instead of data, some commands request action to occur on an axis or group of axes. For example, ST AB stops motion on both the A and B axes. Commas are not required in this case since the particular axis is specified by the appropriate letter A,B,C or D. If no parameters follow the instruction, action will take place on all axes. Here are some examples of syntax for requesting action:

BG A	Begin A only
BG B	Begin B only
BG ABCD	Begin all axes
BG BD	Begin B and D only
BG	Begin all axes

For systems with 5 or more axes, the axes are referred to as A,B,C,D,E,F,G,H.

BG ABCDEFGH	Begin all axes
BG D	Begin D only

Command Syntax - Binary

Some commands have an equivalent binary value. In the command reference, these values are listed next to the command in parentheses in hexadecimal format *. Binary communication mode can be executed much faster than ASCII commands. Binary format can only be used when commands are sent from the PC and cannot be embedded in an application program.

* hexadecimal format represents a byte as two 4 bit values. Each 4 bit value is represented as a single character with a decimal equivalent between 0 and 15. The characters used for representing 10-15 is A,B,C,D,E and F. For example, the hexadecimal value **6D** represent the binary value 01101101. Negative values are represented in 2's complement.

Binary Command Format

All binary commands have a 4 byte header and is followed by data fields. The 4 bytes are specified in hexadecimal format.

Header Format:

Byte 1

Specifies the command number between 80 to FF. The complete binary command number table is listed below.

Byte 2

Specifies the # of bytes in each field as 0,1,2,4 or 6 as follows:

00	No datafields (i.e. SH or BG)
01	One byte per field
02	One word (2 bytes per field)
04	One long word (4 bytes) per field
06	Galil real format (4 bytes integer and 2 bytes fraction)

Byte 3

00	No coordinated motion

Byte 4

Specifies the axis # or data field as follows

Bit 7 = H axis or 8th data field

Bit 6 = G axis or 7th data field

Bit 5 = F axis or 6th data field

Bit 4 = E axis or 5th data field

Bit 3 = D axis or 4th data field

Bit 2 = C axis or 3rd data field

Bit 1 = B axis or 2nd data field

Bit 0 = A axis or 1st data field

Datafields Format

Datafields must be consistent with the format byte and the axes byte. For example, the command PR 1000 would be

A7 02 00 01 03 ES

where A7 is the command number for PR

02 specifies 2 bytes for each data field

00 S is not active for PR

00 specifies bit 0 is active for A axis (2^0=1)

03 E8 represents 1000

FE OE represents -500

Example

The command ST Awould be

A1 00 00 01

where A1 is the command number for ST

00 specifies 0 data fields

00 specifies no coordinated motion

01 specifies stop X (bit 0), (2^0=1)

Binary command table

COMMAND	NO.	COMMAND	NO.	COMMAND	NO.
reserved	80	reserved	ab	reserved	d6
KP	81	reserved	ac	reserved	d7
KI	82	reserved	ad	RP	d8
KD	83	reserved	ae	TP	d9
DV	84	reserved	af	TE	da
AF	85	reserved	b0	TD	db
reserved	86	reserved	b1	TV	dc
PL	87	reserved	b2	RL	dd
ER	88	reserved	b3	TT	de
IL	89	reserved	b4	TS	df
TL	8a	reserved	b5	TI	e0
MT	8b	reserved	b6	SC	e1
CE	8c	reserved	b7	reserved	e2
OE	8d	reserved	b8	reserved	e3
FL	8e	reserved	b9	CO	e4
BL	8f	reserved	ba	TM	e5
AC	90	reserved	bb	CN	e6
DC	91	reserved	bc	LZ	e7
SP	92	CM	bd	OP	e8
IT	93	CD	be	OB	e9

FA	94	DT	bf	SB	ea
FV	95	reserved	c0	CB	eb
GR	96	reserved	c1	II	ec
DP	97	reserved	c2	reserved	ed
DE	98	reserved	c3	AL	ee
OF	99	reserved	c4	reserved	ef
reserved	9a	reserved	c5	reserved	f0
reserved	9b	reserved	c6	reserved	f1
reserved	9c	reserved	c7	reserved	f2
reserved	9d	AM	c8	reserved	f3
reserved	9e	MC	c9	reserved	f4
reserved	9f	TW	ca	reserved	f5
BG	a0	MF	cb	reserved	f6
ST	a1	MR	cc	reserved	f7
AB	a2	AD	cd	reserved	f8
HM	a3	AP	ce	reserved	f9
FE	a4	AR	cf	reserved	fa
FI	a5	AS	d0	reserved	fb
PA	a6	AI	d1	reserved	fc
PR	a7	AT	d2	reserved	fd
JG	a8	WT	d3	reserved	fe
MO	a9	WC	d4	reserved	ff
SH	aa	reserved	d5		

Controller Response to DATA

The CDS-3310 returns a : for valid commands and a ? for invalid commands.

For example, if the command BG is sent in lower case, the CDS-3310 will return a ?.

:bg <return>	invalid command, lower case
?	CDS-3310 returns a ?

When the controller receives an invalid command the user can request the error code. The error code will specify the reason for the invalid command response. To request the error code type the command: TC1 For example:

?TC1 <return>	Tell Code command
1 Unrecognized Command	Returned response

There are many reasons for receiving an invalid command response. The most common reasons are: unrecognized command (such as typographical entry or lower case), command given at improper time (such as during motion), or a command out of range (such as exceeding maximum speed). A complete listing of all codes is listed in the TC command in the Command Reference.

Interrogating the Controller

Interrogation Commands

The CDS-3310 has a set of commands that directly interrogate the controller. When the command is entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), Variable Format (VF) and Leading Zeros (LZ) command. See Chapter 7 and the Command Reference.

Summary of Interrogation Commands

RP	Report Command Position
RL	Report Latch
^R ^V	Firmware Revision Information
SC	Stop Code
TB	Tell Status
TC	Tell Error Code
TD	Tell Dual Encoder
TE	Tell Error
TH	Tell Handle Information
TI	Tell Input
TP	Tell Position
TR	Trace
TS	Tell Switches
TT	Tell Torque
TV	Tell Velocity
TH	Tell Ethernet Handle status
TZ	Tell I/O Status

For example, the following example illustrates how to display the current position of the X axis:

TP A <return>	Tell position A
0000000000	Controllers Response
TP AB <return>	Tell position A and B
0000000000,0000000000	Controllers Response

Interrogating Current Commanded Values

Most commands can be interrogated by using a question mark (?) as the axis specifier. Type the command followed by a ? for each axis requested.

PR ?,?,?,?	Request A,B,C,D values
PR ,?	Request B value only

The controller can also be interrogated with operands.

Operands

Most CDS-3310 commands have corresponding operands that can be used for interrogation. Operands must be used inside of valid DMC expressions. For example, to display the value of an operand, the user could use the command:

MG ‘operand’ where ‘operand’ is a valid DMC operand

All of the command operands begin with the underscore character (_). For example, the value of the current position on the A axis can be assigned to the variable ‘V’ with the command:

V=_TPA

The Command Reference denotes all commands which have an equivalent operand as "Used as an Operand". Also, see description of operands in Chapter 7.

Chapter 6 Programming Motion

Overview

The CDS-3310 provides several modes of motion, including independent positioning and jogging, and electronic gearing. Each one of these modes is discussed in the following sections.

The example applications described below will help guide you to the appropriate mode of motion.

Example Application	Mode of Motion	Commands
Absolute or relative positioning where each axis is independent and follows prescribed velocity profile.	Independent Axis Positioning	PA,PR SP,AC,DC
Velocity control where no final endpoint is prescribed. Motion stops on Stop command.	Independent Jogging	JG AC,DC ST
Absolute positioning mode where absolute position targets may be sent to the controller while the axis is in motion.	Position Tracking	PA PT SP AC, DC
Motion Path described as incremental position points versus time.	Contour Mode	CM CD DT WC
Electronic gearing where slave axes are scaled to master axis which can move in both directions.	Electronic Gearing	GA GD _GP GR
Master/slave where slave axes must follow a master such as conveyer speed.	Electronic Gearing	GA GD _GP GR
Moving along arbitrary profiles or mathematically prescribed profiles such as sine or cosine trajectories.	Contour Mode	CM CD DT WC

Teaching or Record and Play Back	Contour Mode with Automatic Array Capture	CM CD DT WC RA RD RC
Backlash Correction	Dual Loop	DV
Smooth motion while operating in independent axis positioning	Independent Motion Smoothing	IT

Independent Axis Positioning

In this mode, motion between the specified axes is independent, and each axis follows its own profile. The user specifies the desired absolute position (PA) or relative position (PR), slew speed (SP), acceleration ramp (AC), and deceleration ramp (DC), for each axis. On begin (BG), the CDS-3310 profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every second sample period until the specified profile is complete. Motion is complete when the last position command is sent by the CDS-3310 profiler.

NOTE: The actual motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.

The Begin (BG) command can be issued for all axes either simultaneously or independently. ABC or D axis specifiers are required to select the axes for motion. When no axes are specified, this causes motion to begin on all axes.

The speed (SP) and the acceleration (AC) can be changed at any time during motion, however, the deceleration (DC) and position (PR or PA) cannot be changed until motion is complete. Remember, motion is complete when the profiler is finished, not when the actual motor is in position. The Stop command (ST) can be issued at any time to decelerate the motor to a stop before it reaches its final position.

An incremental position movement (IP) may be specified during motion as long as the additional move is in the same direction. Here, the user specifies the desired position increment, n. The new target is equal to the old target plus the increment, n. Upon receiving the IP command, a revised profile will be generated for motion towards the new end position. The IP command does not require a BG.

NOTE: If the motor is not moving, the IP command is equivalent to the PR and BG command combination.

Command Summary - Independent Axis

COMMAND	DESCRIPTION
PR A,B,C,D	Specifies relative distance
PA A,B,C,D	Specifies absolute position
SP A,B,C,D	Specifies slew speed
AC A,B,C,D	Specifies acceleration rate
DC A,B,C,D	Specifies deceleration rate
BG ABCD	Starts motion
ST ABCD	Stops motion before end of move

IP A,B,C,D	Changes position target
IT A,B,C,D	Time constant for independent motion smoothing
AM ABCD	Trippoint for profiler complete
MC ABCD	Trippoint for "in position"

The CDS-3310 also allows use of single axis specifiers such as PRB=2000

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACx	Return acceleration rate for the axis specified by 'x'
_DCx	Return deceleration rate for the axis specified by 'x'
_SPx	Returns the speed for the axis specified by 'x'
_PAx	Returns current destination if 'x' axis is moving, otherwise returns the current commanded position if in a move.
_PRx	Returns current incremental distance specified for the 'x' axis

Examples

Absolute Position Movement

Instruction	Interpretation
PA 10000,20000	Specify absolute A,B position
AC 1000000,1000000	Acceleration for A,B
DC 1000000,1000000	Deceleration for A,B
SP 50000,30000	Speeds for A,B
BG AB	Begin motion

Multiple Move Sequence

Required Motion Profiles:

A-Axis	500 counts	Position
	10000 count/sec	Speed
	500000 counts/sec^2	Acceleration
B-Axis	1000 counts	Position
	15000 count/sec	Speed
	500000 counts/sec^2	Acceleration
C-Axis	100 counts	Position
	5000 counts/sec	Speed
	500000 counts/sec	Acceleration

This example will specify a relative position movement on A, B and C axes. The movement on each axis will be separated by 20 msec. Fig. 6.1 shows the velocity profiles for the A,B and C axis.

Instruction	Interpretation
#A	Begin Program
PR 2000,500,100	Specify relative position movement of 2000, 500 and 100 counts for A,B and C axes.
SP 15000,10000,5000	Specify speed of 10000, 15000, and 5000 counts / sec
AC 500000,500000,500000	Specify acceleration of 500000 counts / sec^2 for all axes
DC 500000,500000,500000	Specify deceleration of 500000 counts / sec^2 for all axes
BG A	Begin motion on the A axis
WT 20	Wait 20 msec
BG B	Begin motion on the B axis
WT 20	Wait 20 msec
BG C	Begin motion on C axis
EN	End Program

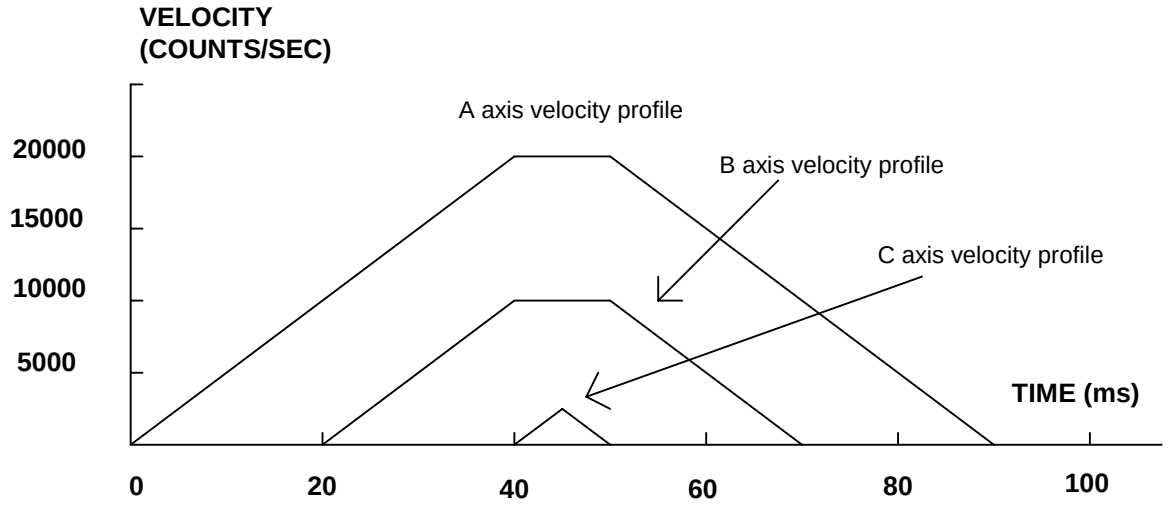


Figure 6.1 - Velocity Profiles of ABC

Notes on fig 6.1: The A and B axis have a 'trapezoidal' velocity profile, while the C axis has a 'triangular' velocity profile. The A and B axes accelerate to the specified speed, move at this constant speed, and then decelerate such that the final position agrees with the command position, PR. The C axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position. All 3 axes have the same acceleration and deceleration rate, hence, the slope of the rising and falling edges of all 3 velocity profiles are the same.

Independent Jogging

The jog mode of motion is very flexible because speed, direction and acceleration can be changed during motion. The user specifies the jog speed (JG), acceleration (AC), and the deceleration (DC) rate for each axis. The direction of motion is specified by the sign of the JG parameters. When the begin command is given (BG), the motor accelerates up to speed and continues to jog at that speed until a new speed or stop (ST) command is issued. If the jog speed is changed during motion, the controller will make a accelerated (or decelerated) change to the new speed.

An instant change to the motor position can be made with the use of the IP command. Upon receiving this command, the controller commands the motor to a position which is equal to the specified

increment plus the current position. This command is useful when trying to synchronize the position of two motors while they are moving.

Note that the controller operates as a closed-loop position controller while in the jog mode. The CDS-3310 converts the velocity profile into a position trajectory and a new position target is generated every other sample period. This method of control results in precise speed regulation with phase lock accuracy.

Command Summary - Jogging

COMMAND	DESCRIPTION
AC A,B,C,D	Specifies acceleration rate
BG ABCD	Begins motion
DC A,B,C,D	Specifies deceleration rate
IP A,B,C,D	Increments position instantly
IT A,B,C,D	Time constant for independent motion smoothing
JG +/-A,B,C,D	Specifies jog speed and direction
ST ABCD	Stops motion

Parameters can be set with individual axes specifiers such as JGB=2000 (set jog speed for B axis to 2000).

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACx	Return acceleration rate for the axis specified by 'x'
_DCx	Return deceleration rate for the axis specified by 'x'
_SPx	Returns the jog speed for the axis specified by 'x'
_TVx	Returns the actual velocity of the axis specified by 'x' (averaged over .25 sec)

Examples

Jog in X only

Jog A motor at 50000 count/s. After A motor is at its jog speed, begin jogging C in reverse direction at 25000 count/s.

Instruction	Interpretation
#A	Label
AC 20000,,20000	Specify A,C acceleration of 20000 cts / sec
DC 20000,,20000	Specify A,C deceleration of 20000 cts / sec
JG 50000,,-25000	Specify jog speed and direction for A and C axis
BG A	Begin A motion
AS A	Wait until A is at speed
BG C	Begin C motion
EN	

Position Tracking

The Galil controller may be placed in the position tracking mode to support changing the target of an absolute position move on the fly. New targets may be given in the same direction or the opposite direction of the current position target. The controller will then calculate a new trajectory based upon the new target and the acceleration, deceleration, and speed parameters that have been set. The motion profile in this mode is trapezoidal. There is not a set limit governing the rate at which the end point may be changed, however at the standard TM rate, the controller updates the position information at the rate of 1msec. The controller generates a profiled point every other sample, and linearly interpolates one sample between each profiled point. Some examples of applications that may use this mode are satellite tracking, missile tracking, random pattern polishing of mirrors or lenses, or any application that requires the ability to change the endpoint without completing the previous move.

The PA command is typically used to command an axis or multiple axes to a specific absolute position. For some applications such as tracking an object, the controller must proceed towards a target and have the ability to change the target during the move. In a tracking application, this could occur at any time during the move or at regularly scheduled intervals. For example if a robot was designed to follow a moving object at a specified distance and the path of the object wasn't known the robot would be required to constantly monitor the motion of the object that it was following. To remain within a specified distance it would also need to constantly update the position target it is moving towards. Galil motion controllers support this type of motion with the position tracking mode. This mode will allow scheduled or random updates to the current position target on the fly. Based on the new target the controller will either continue in the direction it is heading, change the direction it is moving, or decelerate to a stop.

The position tracking mode shouldn't be confused with the contour mode. The contour mode allows the user to generate custom profiles by updating the reference position at a specific time rate. In this mode, the position can be updated randomly or at a fixed time rate, but the velocity profile will always be trapezoidal with the parameters specified by AC, DC, and SP. Updating the position target at a specific rate will not allow the user to create a custom profile.

The following example will demonstrate the possible different motions that may be commanded by the controller in the position tracking mode. In this example, there is a host program that will generate the absolute position targets. The absolute target is determined based on the current information the host program has gathered on the object that it is tracking. The position tracking mode does allow for all of the axes on the controller to be in this mode, but for the sake of discussion, it is assumed that the robot is tracking only in the X dimension.

The controller must be placed in the position tracking mode to allow on the fly absolute position changes. This is performed with the PT command. To place the X axis in this mode, the host would issue PT1 to the controller if both X and Y axes were desired the command would be PT 1,1. The next step is to begin issuing PA command to the controller. The BG command isn't required in this mode, the SP, AC, and DC commands determine the shape of the trapezoidal velocity profile that the controller will use.

Example Motion 1:

The host program determines that the first target for the controller to move to is located at 5000 encoder counts. The acceleration and deceleration should be set to 150,000 cts/sec^2 and the velocity is set to 50,000 cts/sec. The command sequence to perform this is listed below.

COMMAND	DESCRIPTION
PT1	Place the X axis in Position tracking mode
AC150000	Set the X axis acceleration to 150000 cts/sec^2
DC150000	Set the X axis deceleration to 150000 cts/sec^2
SP50000	Set the X axis speed to 50000 cts/sec
PA5000	Command the X axis to absolute position 5000 encoder counts

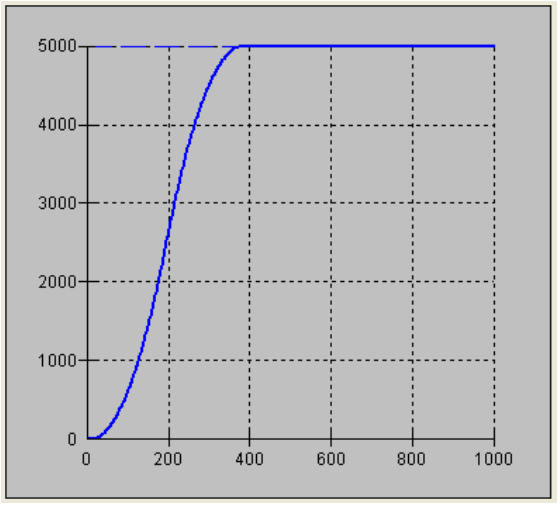


Figure 1 Position vs Time (msec) Motion 1

Example Motion 2:

The previous step showed the plot if the motion continued all the way to 5000, however partway through the motion, the object that was being tracked changed direction, so the host program determined that the actual target position should be 2000 cts at that time. Figure 2 shows what the position profile would look like if the move was allowed to complete to 2000 cts. The position was modified when the robot was at a position of 4200 cts. Note that the robot actually travels to a distance of almost 5000 cts before it turns around. This is a function of the deceleration rate set by the DC command. When a direction change is commanded, the controller decelerates at the rate specified by the DC command. The controller then ramps the velocity in up to the value set with SP in the opposite direction traveling to the new specified absolute position. In figure 3 the velocity profile is triangular because the controller doesn't have sufficient time to reach the set speed of 50000 cts/sec before it is commanded to change direction.

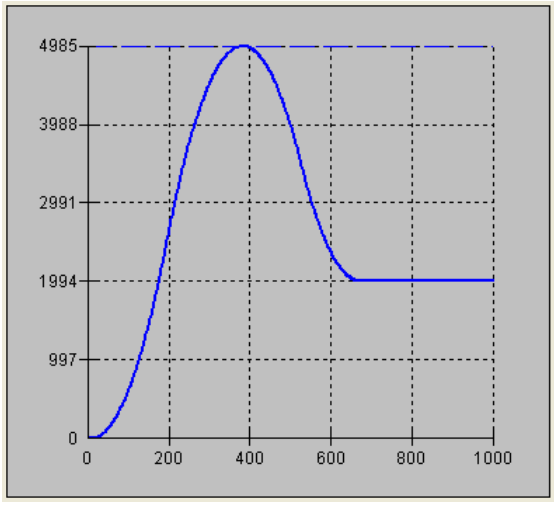


Figure 2: Position vs. Time (msec) Motion 2

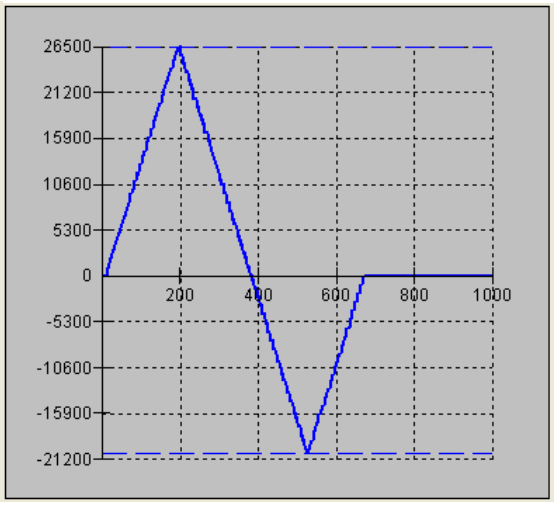


Figure 3 Velocity vs Time (msec) Motion 2

Example Motion 3:

In this motion, the host program commands the controller to begin motion towards position 5000, changes the target to -2000, and then changes it again to 8000. Figure 4 shows the plot of position vs. time, Figure 5 plots velocity vs. time, and Figure 6 demonstrates the use of motion smoothing (IT) on the velocity profile in this mode. The jerk in the system is also affected by the values set for AC and DC.

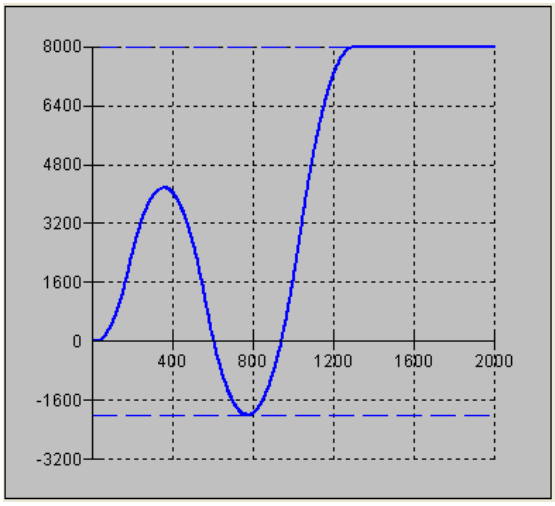


Figure 4 Position vs. Time (msec) Motion 4

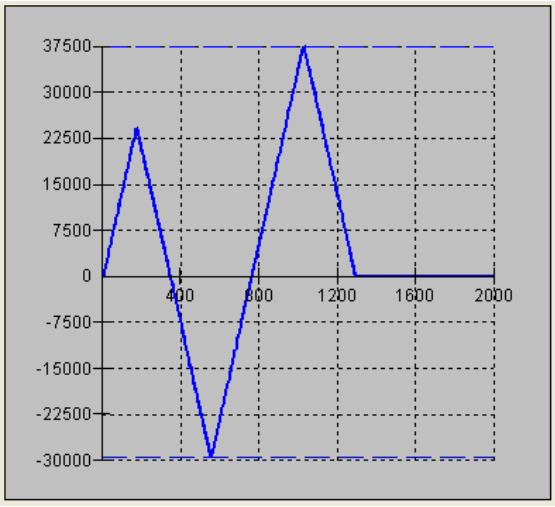


Figure 5 Velocity vs.Time Motion 4

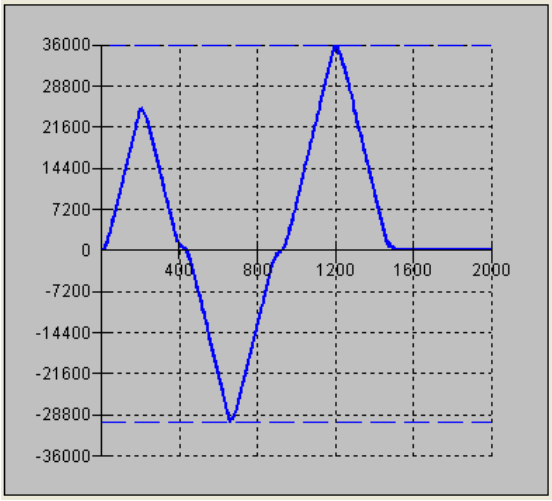


Figure 6 Velocity cts/sec vs. Time (msec) with IT

Note the controller treats the point where the velocity passes through zero as the end of one move, and the beginning of another move. IT is allowed, however it will introduce some time delay.

Trip Points

Most trip points are valid for use while in the position tracking mode. There are a few exceptions to this; the AM and MC commands may not be used while in this mode. It is recommended that MF, MR, or AP be used, as they involve motion in a specified direction, or the passing of a specific absolute position.

Command Summary – Position Tracking Mode

COMMAND	DESCRIPTION
AC n,n,n,n,n,n,n,n	Acceleration settings for the specified axes
AP n,n,n,n,n,n,n,n	Trip point that holds up program execution until an absolute position has been reached
DC n,n,n,n,n,n,n,n	Deceleration settings for the specified axes
MF n,n,n,n,n,n,n,n	Trip point to hold up program execution until n number of counts have passed in the forward direction. Only one axis at a time may be specified.
MR n,n,n,n,n,n,n,n	Trip point to hold up program execution until n number of counts have passed in the reverse direction. Only one axis at a time may be specified.
PT n,n,n,n,n,n,n,n	Command used to enter and exit the Trajectory Modification Mode
PA n,n,n,n,n,n,n,n	Command Used to specify the absolute position target
SP n,n,n,n,n,n,n,n	Command used to enter and exit the Trajectory Modification Mode

Electronic Gearing

This mode allows the axis to be electronically geared to the auxiliary encoder. The masters may rotate in both directions and the geared axis will follow at the specified gear ratio. The gear ratio may be changed during motion.

The command GA DA specifies the master axes. GR a specifies the gear ratio for the slave where the ratio may be a number between +/-127.9999 with a fractional resolution of 0.0001. GR 0 turns off gearing. A limit switch or ST command disables gearing. GR causes the axis to be geared to the actual position of the master.

Electronic gearing allows the geared motor to perform a second independent or coordinated move in addition to the gearing. For example, when a geared motor follows a master at a ratio of 1:1, it may be advanced an additional distance with PR, or JG.

Ramped Gearing

In some applications, especially when the master is traveling at high speeds, it is desirable to have the gear ratio ramp gradually to minimize large changes in velocity on the slave axis when the gearing is engaged. For example if the master axis is already traveling at 1,000,000 cts/sec and the slave will be geared at a ratio of 1:1 when the gearing is engaged, the slave will instantly develop following error, and command maximum current to the motor. This can be a large shock to the system. For many applications it is acceptable to slowly ramp the engagement of gearing over a greater time frame. Galil allows the user to specify an interval of the master axis over which the gearing will be engaged. For example, the same master X axis in this case travels at 1,000,000 counts/sec, and the gear ratio is 1:1, but the gearing is slowly engaged over 30,000 cts of the master axis, greatly diminishing the initial shock to the slave axis. Figure 1 below shows the velocity vs. time profile for instantaneous gearing. Figure 2 shows the velocity vs. time profile for the gradual gearing engagement.

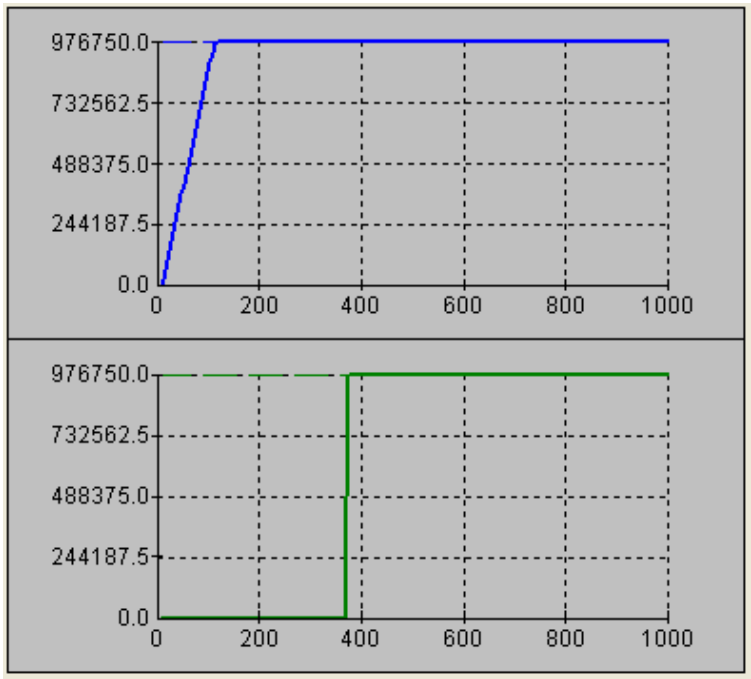


Figure 1 Velocity cts/sec vs. Time (msec) Instantaneous Gearing Engagement

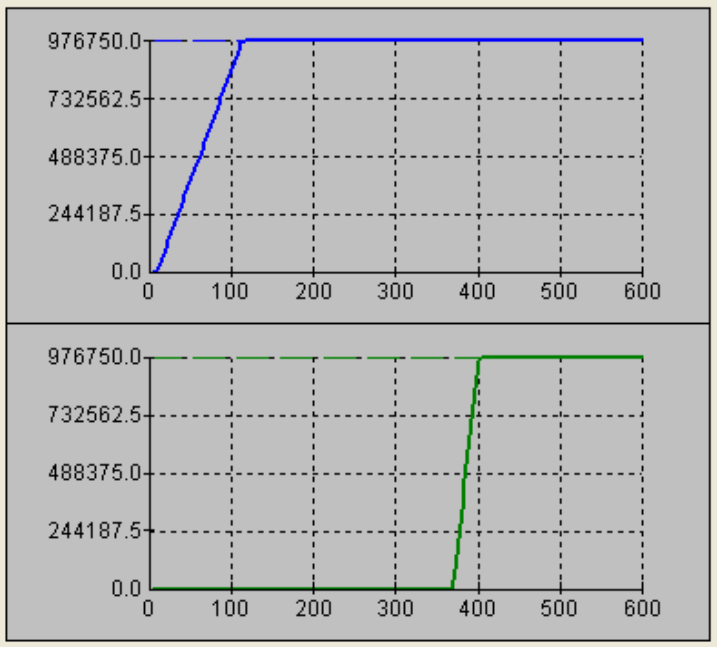


Figure 2 Velocity (cts/sec) vs. Time (msec) Ramped Gearing

The slave axis for each figure is shown in the bottom portion of the figure; the master axis is shown in the top portion. The shock to the slave axis will be significantly less in figure 2 than in figure1. The ramped gearing does have one consequence. There isn't a true synchronization of the two axes, until the gearing ramp is complete. The slave will lag behind the true ratio during the ramp period. If exact position synchronization is required from the point gearing is initiated, then the position must be commanded in addition to the gearing. The controller keeps track of this position phase lag with the _GP operand. The following example will demonstrate how the command is used.

Example – Electronic Gearing Over a Specified Interval

Objective: Run a geared motor at a speed of 1.132 times the speed of an external master. Because the master is traveling at high speeds, it is desirable for the speeds to change slowly. We will implement the gearing change over 6000 counts (3 revolutions) of the master axis.

GA DA	Specify aux encoder as the master
GD 6000	Specify ramped gearing over 6000 counts of the master axis.
GR 1.132	Specify gear ratios

Question: What is the effect of the ramped gearing?

Answer: Had GD not been issued, gearing would take effect immediately and if the master traveled 6000 counts, the slave would travel 6792 counts. Using the ramped gearing, the slave will engage gearing gradually. Since the gearing is engaged over the interval of 6000 counts of the master, the slave will only travel ~3396 counts. The difference between these two values is stored in the _GPn operand. If exact position synchronization is required, the IP command is used to adjust for the difference.

Command Summary - Electronic Gearing

COMMAND	DESCRIPTION
GA n	Specifies master axes for gearing where n = DX for auxiliary encoder
GD a	Sets the distance the master will travel for the gearing change to take full effect.
_GPn	This operand keeps track of the difference between the theoretical distance traveled if gearing changes took effect immediately, and the distance traveled since gearing changes take effect over a specified interval.
GR a	Sets gear ratio. 0 disables electronic gearing for specified axis.

Examples

Electronic Gearing

Objective: Run a geared motors at a speed of 1.132 times the speed of an external master. The master is driven at speeds between 0 and 1800 RPM (2000 counts/rev encoder).

Solution:

Instruction	Interpretation
GA DA	Specify aux encoder as the master
GR 1.132	Specify gear ratios

Now suppose the gear ratio of the A-axis is to change on-the-fly to 2. This can be achieved by commanding:

GR 2	Specify gear ratio for A axis to be 2

Phase Corrections

You may also perform profiled position corrections in the electronic gearing mode. Suppose, for example, that you need to advance the slave 10 counts. Simply command

IP 10	Specify an incremental position movement of 10

Under these conditions, this IP command is equivalent to:

PR10	Specify position relative movement of 10
BGA	Begin motion

Often the correction is quite large. Such requirements are common when synchronizing cutting knives or conveyor belts.

Synchronize two conveyor belts with trapezoidal velocity correction.

Instruction	Interpretation
GA DA	Define aux encoder as the master axis for B.
GR2	Set gear ratio 2:1
PR300	Specify correction distance
SP5000	Specify correction speed
AC100000	Specify correction acceleration
DC100000	Specify correction deceleration
BG	Start correction

Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization of the axis of motion. Similar to the gearing mode, the CDS-3310 use the auxiliary encoder or the imaginary axis as the master axis.

The electronic cam is a more general type of electronic gearing which allows a table-based relationship between the axes. It allows synchronizing of the motor based on an auxiliary encoder input, or on an internal reference from the imaginary axis, n.

To illustrate the procedure of setting the cam mode, consider the cam relationship shown in Figure 6.4.

Step 1. Selecting the master axis.

The first step in the electronic cam mode is to select the master axis. This is done with the instruction

EAp where p = X,N

p is the selected master axis

In this example x axis will be the master. Thus we specify EAX

Step 2. Specify the master cycle and the change in the slave axes.

In the electronic cam mode, the position of the master is always expressed within one cycle. In this example, the position of x is always expressed in the range between 0 and 6000. Similarly, the slave position is also redefined such that it starts at zero and ends at 1500. At the end of a cycle when the master is 6000 and the slave is 1500, the positions of both x and y are redefined as zero. To specify the master cycle and the slave cycle change, we use the instruction EM.

EM x,y

where x,y specify the cycle of the master and the total change of the slave over one cycle. On the CDS-3310, x will always be the master cycle, and y will be the slave cycle

The cycle of the master is limited to 8,388,607 whereas the slave change per cycle is limited to 2,147,483,647. If the change is a negative number, the absolute value is specified. For the given example, the cycle of the master is 6000 counts and the change in the slave is 1500. Therefore, we use the instruction:

EM 6000,1500

Step 3. Specify the master interval and starting point.

Next we need to construct the ECAM table. The table is specified at uniform intervals of master positions. Up to 256 intervals are allowed. The size of the master interval and the starting point are specified by the instruction:

EP m,n

where m is the interval width in counts, and n is the starting point.

For the given example, we can specify the table by specifying the position at the master points of 0, 2000, 4000 and 6000. We can specify that by

```
EP 2000,0
```

Step 4. Specify the slave positions.

Next, we specify the slave positions with the instruction

```
ET[n]=x
```

where n indicates the order of the point.

The value, n, starts at zero and may go up to 256. The parameter x indicates the corresponding slave position. For this example, the table may be specified by

ET[0]=0

ET[1]=3000

ET[2]=2250

ET[3]=1500

This specifies the ECAM table.

Step 5. Enable the ECAM

To enable the ECAM mode, use the command

```
EB n
```

where n=1 enables ECAM mode and n=0 disables ECAM mode.

Step 6. Engage the slave motion

To engage the slave motion, use the instruction

```
EG x
```

where x is the master position at which the x-axis must be engaged.

If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the slave position is redefined, modulo one cycle.

Step 7. Disengage the slave motion

To disengage the cam, use the command

EQ x

where x is the master position at which the x-axis must be disengaged.

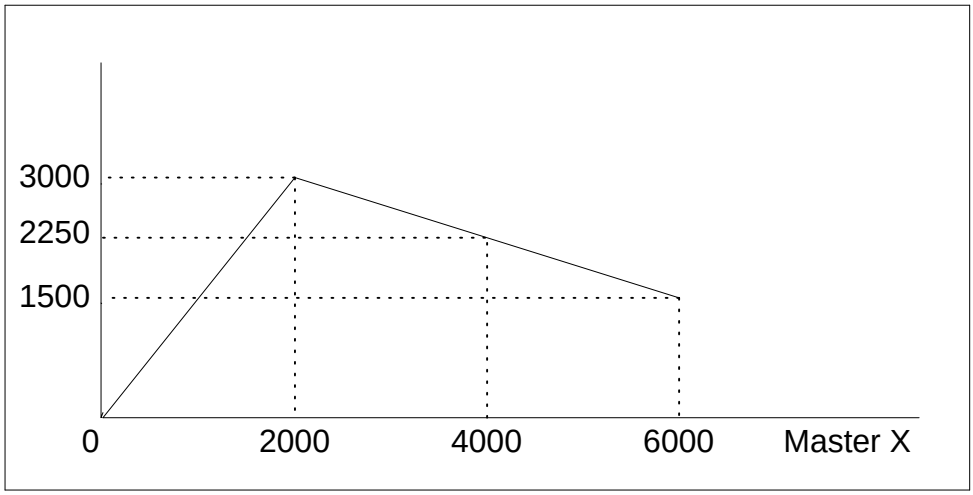


Figure 6.4 - Electronic Cam Example

This disengages the slave axis at a specified master position. If the parameter is outside the master cycle, the stopping is instantaneous.

To illustrate the complete process, consider the cam relationship described by

the equation:

$$X = 0.5 * N + 100 \sin (0.18 * N)$$

where N is the master, with a cycle of 2000 counts.

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up.

The instruction EAN defines N as the master axis. The cycle of the master is 2000. Over that cycle, X varies by 1000. This leads to the instruction EM 2000,1000.

Suppose we want to define a table with 100 segments. This implies increments of 20 counts each. If the master points are to start at zero, the required instruction is EP 20,0.

The following routine computes the table points. As the phase equals 0.18N and N varies in increments of 20, the phase varies by increments of 3.6°. The program then computes the values of X according to the equation and assigns the values to the table with the instruction ET[n] =X.

Instruction	**Interpretation**
#SETUP	Label
EAN	Select N as master
EM 2000,1000	Cam cycles
EP 20,0	Master position increments
n = 0	Index
#LOOP	Loop to construct table from equation
P = n∗3.6	Note 3.6 = 0.18∗20
S = @SIN [P] ∗100	Define sine position
X = n ∗10+S	Define slave position
ET [n] =X	Define table
n = n+1	
JP #LOOP, n<=100	Repeat the process
EN	

Now suppose that the slave axis is engaged with a start signal, input 1, but that both the engagement and disengagement points must be done at the center of the cycle: N = 1000 and X = 500. This implies that X must be driven to that point to avoid a jump.

This is done with the program:

Instruction	**Interpretation**
#RUN	Label
EB1	Enable cam
PA500	X starting position
SP5000	X speed
BGY	Move X motor
AM	After X moved
AI1	Wait for start signal
EG1000	Engage slave
AI – 1	Wait for stop signal
EQ1000	Disengage slave
EN	End

The following example illustrates a cam program with a master axis, N.

Instruction	**Interpretation**
#A;V1=0	Label; Initialize variable

PA 0;BGX;AMX	Go to position 0,0 on X axis
EA N	N axis as the Master for ECAM
EM 4000,0	Change for N is 4000, zero for X
EP400,0	ECAM interval is 400 counts with zero start
ET[0]=0	When master is at 0 position; 1st point.
ET[1]=20	2nd point in the ECAM table
ET[2]=60	3rd point in the ECAM table
ET[3]=120	4th point in the ECAM table
ET[4]=140	5th point in the ECAM table
ET[5]=140	6th point in the ECAM table
ET[6]=140	7th point in the ECAM table
ET[7]=120	8th point in the ECAM table
ET[8]=60	9th point in the ECAM table
ET[9]=20	10th point in the ECAM table
ET[10]=0	Starting point for next cycle
EB 1	Enable ECAM mode
JGN=4000	Set N to jog at 4000
EG 0	Engage X when Master = 0
BGX	Begin jog on N axis
#LOOP;JP#LOOP,V1=0	Loop until the variable is set
EQ,2000	Disengage X when Master = 2000
MF2000	Wait until the Master goes to 2000
ST N	Stop the N axis motion
EB 0	Exit the ECAM mode
EN	End of the program

Contour Mode

The CDS-3310 also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed for 1 to 8 axes. This is ideal for following computer generated paths such as parabolic, spherical or user-defined profiles. The path is not limited to straight line and arc segments and the path length may be infinite.

Specifying Contour Segments

The Contour Mode is specified with the command, CM. For example, CMAC specifies contouring on the A and C axes. Any axes that are not being used in the contouring mode may be operated in other modes.

A contour is described by position increments which are described with the command, CD a,b,c,d over a time interval, DT n. The parameter, n, specifies the time interval. The time interval is defined as 2^n ms, where n is a number between 1 and 8. The controller performs linear interpolation between the specified increments, where one point is generated for each millisecond.

Consider, for example, the trajectory shown in Fig. 6.6. The position A may be described by the points:

Point 1 A=0 at T=0ms

Point 2	A=48 at T=4ms
Point 3	A=288 at T=12ms
Point 4	A=336 at T=28ms

The same trajectory may be represented by the increments

Increment 1	DA=48	Time=4	DT=2
Increment 2	DA=240	Time=8	DT=3
Increment 3	DA=48	Time=16	DT=4

When the controller receives the command to generate a trajectory along these points, it interpolates linearly between the points. The resulting interpolated points include the position 12 at 1 msec, position 24 at 2 msec, etc.

The programmed commands to specify the above example are:

Instruction	Interpretation
#A	Label
CMA	Specifies A axis for contour mode
DT 2	Specifies first time interval, 2^2 ms
CD 48;WC	Specifies first position increment
DT 3	Specifies second time interval, 2^3 ms
CD 240;WC	Specifies second position increment
DT 4	Specifies the third time interval, 2^4 ms
CD 48;WC	Specifies the third position increment
DT0;CD0	Exits contour mode
EN	

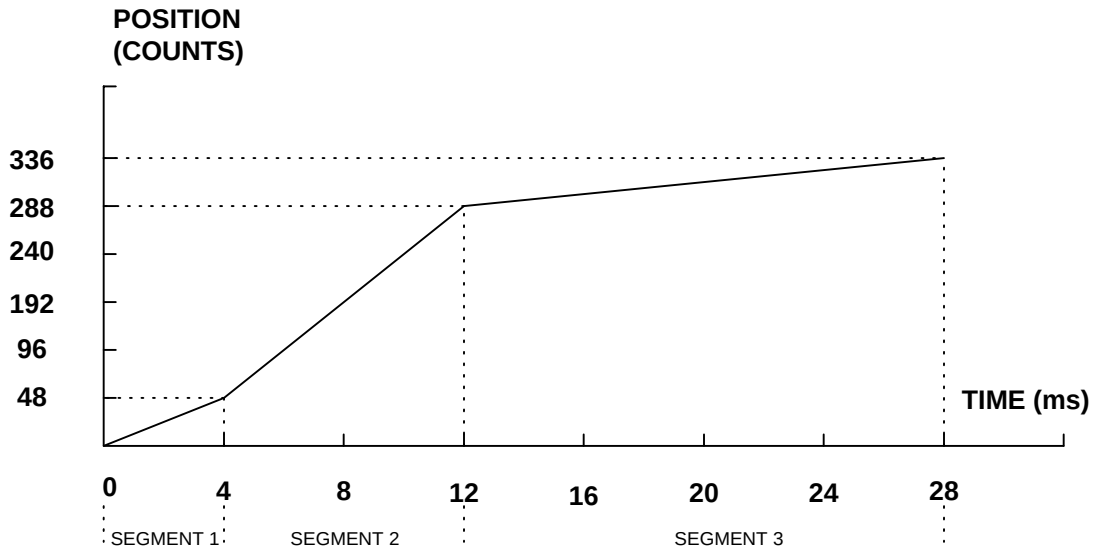


Figure 6.6 - The Required Trajectory

Additional Commands

The command, WC, is used as a trippoint "Wait for Contour". This allows the CDS-3310 to use the next increment only when it is finished with the previous one. Zero parameters for DT followed by zero parameters for CD exit the contour mode.

If no new data record is found and the controller is still in the contour mode, the controller waits for new data. No new motion commands are generated while waiting. If bad data is received, the controller responds with a ?.

Command Summary - Contour Mode

COMMAND	DESCRIPTION
CM ABCDEFGH	Specifies which axes for contouring mode. Any non-contouring axes may be operated in other modes.
CD a,b,c,d,e,f,g,h	Specifies position increment over time interval. Range is +/-32,000. (Zero ends contour mode, when issued following DT0)
DT n	Specifies time interval 2^n msec for position increment, where n is an integer between 1 and 8. Zero ends contour mode. If n does not change, it does not need to be specified with each CD.
WC	Waits for previous time interval to be complete before next data record is processed.

General Velocity Profiles

The Contour Mode is ideal for generating any arbitrary velocity profiles. The velocity profile can be specified as a mathematical function or as a collection of points.

The design includes two parts: Generating an array with data points and running the program.

Example

Generating an Array

Consider the velocity and position profiles shown in Fig. 6.7. The objective is to rotate a motor a distance of 6000 counts in 120 ms. The velocity profile is sinusoidal to reduce the jerk and the system vibration. If we describe the position displacement in terms of A counts in B milliseconds, we can describe the motion in the following manner:

$$\omega = \frac{\mathrm{A}}{\mathrm{B}}\left(1 - \cos(2\pi \mathrm{T}/\mathrm{B})\right)$$

$$\mathrm{X} = \frac{AT}{B} - \frac{A}{2\pi}\sin(2\pi \mathrm{T}/\mathrm{B})$$

NOTE: ω is the angular velocity; A is the position; and T is the variable, time, in milliseconds.

In the given example, A=6000 and B=120, the position and velocity profiles are:

$$A = 50T - (6000/2\pi) \sin (2\pi\ T/120)$$

Note that the velocity, ω, in count/ms, is

$$\omega = 50\ [1 - \cos 2\pi\ T/120]$$

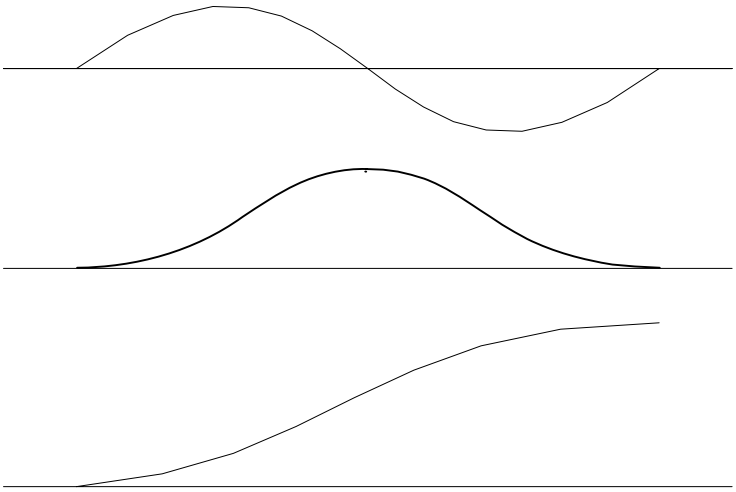


Figure 6.7 - Velocity Profile with Sinusoidal Acceleration

The CDS-3310 can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for A is written as:

A = 50T - 955 sin 3T

A complete program to generate the contour movement in this example is given below. To generate an array, we compute the position value at intervals of 8 ms. This is stored at the array **pos**. Then, the difference between the positions is computed and is stored in the array **dir**. Finally the motors are run in the contour mode.

```
#POINTS                  Program defines A points
DM pos[16]               Allocate memory
DM dir[15]
c=0;d=0                  Set initial conditions, C is index
t=0                      T is time in ms
#A
v1=50*T
v2=3*T                   Argument in degrees
v3=-955*@SIN[V2]+V1      Compute position
v4=@INT[V3]              Integer value of V3
pos[c]=v4                Store in array POS
t=t+8
c=c+1
JP #A,c<16
#B                       Program to find position differences
c=0
#c
d=c+1
dir[c]=pos [d]- pos [c]  Compute the difference and store
c=c+1
JP #c,c<15
```

EN	End first program
#RUN	Program to run motor
CMA	Contour Mode
DT3	8 millisecond intervals
c=0	
#E	
CD dir[c]	Contour Distance is in dir
WC	Wait for completion
c=c+1	
JP #E,c<15	
DT0	
CD0	Stop Contour
EN	End the program

Teach (Record and Play-Back)

Several applications require teaching the machine a motion trajectory. Teaching can be accomplished using the CDS-3310 automatic array capture feature to capture position data. The captured data may then be played back in the contour mode. The following array commands are used:

DM C[n]	Dimension array
RA C[]	Specify array for automatic record
RD _TPA	Specify data for capturing (such as _TPA or _TPC)
RC n,m	Specify capture time interval where n is 2^n samples, m is number of records to be captured
RC? or _RC	Returns a 1 if recording

Record and Playback Example

Instruction	**Interpretation**
#RECORD	Begin Program
DP0	Define position for A axis to be 0
DA*[ ]	De-allocate all arrays
DM xpos [501]	Dimension 501 element array called xpos
RA xpos [ ]	Record Elements into xpos array
RD_TPA	Element to be recorded is encoder position of A axis
MOA	Motor off for A axis
RC2	Begin Recording with a sample rate of 2^2 msec
#LOOP1;JP#LOOP1,_RC=1	Loop until all elements have been recorded
#COMPUTE	Routine to determine the difference between consecutive points
DM dx [500]	Dimension a 500 element array to hold contour points
i = 0	Set loop counter
#LOOP2	Loop to calculate the difference
dx[I]= xpos [i+1]- xpos [i]	Calculate difference
i=i+1	Update loop counter
JP#LOOP2,i<500	Continue looping until dx is full
#PLAYBK	Routine to play back motion that was recorded
SHA	Servo Here

WT1000	Wait 1 sec (1000 msec)
CMA	Specify contour mode on A axis
DT2	Set contour data rate to be 2^2 msec
i=0	Set array index to 0
#LOOP3	Subroutine to execute contour points
CD dx[i];WC	Contour data command; Wait for next contour point
i=i+1	Update index
JP#LOOP3,i<500	Continue until all array elements have been executed
DT0	Set contour update rate to 0
CD0	Disable the contour mode (combination of DT0 and CD0)
EN	End program

For additional information about automatic array capture, see Chapter 7, Arrays.

Virtual Axis

The CDS-3310 controller has an additional virtual axis designated as the N axis. This axis has no encoder and no DAC. However, it can be commanded by the commands:

AC, DC, JG, SP, PR, PA, BG, IT, GA, ST, DP, RP

Dual Loop (Auxiliary Encoder)

The CDS-3310 provides an interface for a second encoder except when configured for output compare. When used, the second encoder is typically mounted on the motor or the load, but may be mounted in any position. The most common use for the second encoder is backlash compensation, described below.

The second encoder may be a standard quadrature type, or it may provide pulse and direction. The controller also offers the provision for inverting the direction of the encoder rotation. The main and the auxiliary encoders are configured with the CE command. The command form is CE a,b,c,d (or a,b,c,d,e,f,g,h for controllers with more than 4 axes) where the parameters a,b,c,d each equal the sum of two integers m and n. m configures the main encoder and n configures the auxiliary encoder.

Using the CE Command

m=	Main Encoder	n=	Second Encoder
0	Normal quadrature	0	Normal quadrature
1	Pulse & direction	4	Pulse & direction
2	Reverse quadrature	8	Reversed quadrature
3	Reverse pulse & direction	12	Reversed pulse & direction

For example, to configure the main encoder for reversed quadrature, m=2, and a second encoder of pulse and direction, n=4, the total is 6, and the command for the A axis is

CE 6

Additional Commands for the Auxiliary Encoder

The command, DE a,b,c,d can be used to define the position of the auxiliary encoders. For example,

DE 0,500,-30,300

sets their initial values.

The positions of the auxiliary encoders may be interrogated with the command, DE?. For example

```
DE ?,,?
```

returns the value of the A and C auxiliary encoders.

The auxiliary encoder position may be assigned to variables with the instructions

```
V1= _DEA
```

The command, TD a,b,c,d, returns the current position of the auxiliary encoder.

The command, DV a,b,c,d, configures the auxilliary encoder to be used for backlash compensation.

Backlash Compensation

There are two methods for backlash compensation using the auxiliary encoders:

1. Continuous dual loop
2. Sampled dual loop

To illustrate the problem, consider a situation in which the coupling between the motor and the load has a backlash. To compensate for the backlash, position encoders are mounted on both the motor and the load.

The continuous dual loop combines the two feedback signals to achieve stability. This method requires careful system tuning, and depends on the magnitude of the backlash. However, once successful, this method compensates for the backlash continuously.

The second method, the sampled dual loop, reads the load encoder only at the end point and performs a correction. This method is independent of the size of the backlash. However, it is effective only in point-to-point motion systems which require position accuracy only at the endpoint.

Example

Continuous Dual Loop

Connect the load encoder to the main encoder port and connect the motor encoder to the dual encoder port. The dual loop method splits the filter function between the two encoders. It applies the KP (proportional) and KI (integral) terms to the position error, based on the load encoder, and applies the KD (derivative) term to the motor encoder. This method results in a stable system.

The dual loop method is activated with the instruction DV (Dual Velocity), where

```
DV      1,1,1,1
```

activates the dual loop for the four axes and

```
DV      0,0,0,0
```

disables the dual loop.

Note that the dual loop compensation depends on the backlash magnitude, and in extreme cases will not stabilize the loop. The proposed compensation procedure is to start with KP=0, KI=0 and to maximize the value of KD under the condition DV1. Once KD is found, increase KP gradually to a maximum value, and finally, increase KI, if necessary.

Sampled Dual Loop

In this example, we consider a linear slide which is run by a rotary motor via a lead screw. Since the lead screw has a backlash, it is necessary to use a linear encoder to monitor the position of the slide. For stability reasons, it is best to use a rotary encoder on the motor.

Connect the rotary encoder to the A-axis and connect the linear encoder to the auxiliary encoder of A. Assume that the required motion distance is one inch, and that this corresponds to 40,000 counts of the rotary encoder and 10,000 counts of the linear encoder.

The design approach is to drive the motor a distance, which corresponds to 40,000 rotary counts. Once the motion is complete, the controller monitors the position of the linear encoder and performs position corrections.

This is done by the following program.

Instruction	Interpretation
#DUALOOP	Label
CE 0	Configure encoder
DE0	Set initial value
PR 40000	Main move
BGA	Start motion
#CORRECT	Correction loop
AMA	Wait for motion completion
v1=10000-_DEA	Find linear encoder error
v2=-_TEA/4+v1	Compensate for motor error
JP#END,@ABS[v2]<2	Exit if error is small
PR v2*4	Correction move
BGA	Start correction
JP#CORRECT	Repeat
#END	
EN	

Motion Smoothing

The CDS-3310 controller allows the smoothing of the velocity profile to reduce the mechanical vibration of the system.

Trapezoidal velocity profiles have acceleration rates which change abruptly from zero to maximum value. The discontinuous acceleration results in jerk which causes vibration. The smoothing of the acceleration profile leads to a continuous acceleration profile and reduces the mechanical shock and vibration.

Using the IT Command:

When operating with servo motors, motion smoothing can be accomplished with the IT command. These commands filter the acceleration and deceleration functions to produce a smooth velocity profile. The resulting velocity profile, has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following command:

IT a,b,c,d	Independent time constant

The command, IT, is used for smoothing independent moves of the type JG, PR, PA.

The smoothing parameters, a,b,c,d and n are numbers between 0 and 1 and determine the degree of filtering. The maximum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Smaller values of the smoothing parameters imply heavier filtering and smoother moves.

The following example illustrates the effect of smoothing. Fig. 6.9 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note that the smoothing process results in longer motion time.

Example

Instruction	Interpretation
PR 20000	Position
AC 100000	Acceleration
DC 100000	Deceleration
SP 5000	Speed
IT .5	Filter for smoothing
BG A	Begin

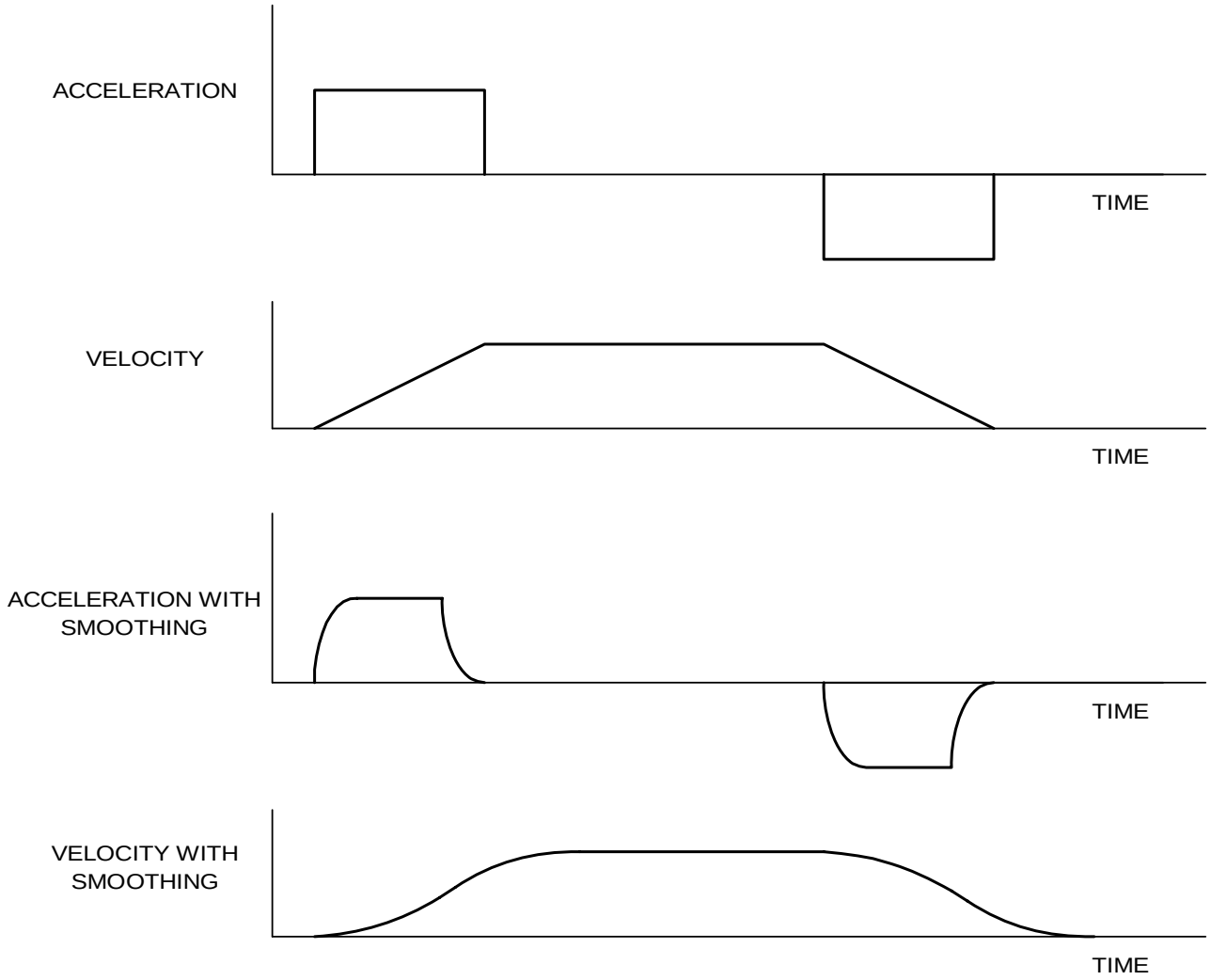


Figure 6.9 - Trapezoidal velocity and smooth velocity profiles

Homing

The Find Edge (FE) and Home (HM) instructions may be used to home the motor to a mechanical reference. This reference is connected to the Home input line. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) is used to define the polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Homing Input. When the Find Edge command and Begin is used, the motor will accelerate up to the slew speed and slew until a transition is detected on the Homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch. The velocity profile generated is shown in Fig. 6.10.

The Home (HM) command can be used to position the motor on the index pulse after the home switch is detected. This allows for finer positioning on initialization. The command sequence HM and BG causes the following sequence of events to occur.

1. Upon begin, motor accelerates to the slew speed. The direction of its motion is determined by the state of the homing input. A zero (GND) will cause the motor to start in the forward direction; +5V will cause it to start in the reverse direction. The CN command is used to define the polarity of the home input.
2. Upon detecting the home switch changing state, the motor begins decelerating to a stop.
3. The motor then traverses very slowly back until the home switch toggles again.
4. The motor then traverses forward until the encoder index pulse is detected.
5. The CDS-3310 defines the home position (0) as the position at which the index was detected.

Example

Instruction	Interpretation
#HOME	Label
AC 1000000	Acceleration Rate
DC 1000000	Deceleration Rate
SP 5000	Speed for Home Search
HM A	Home A
BG A	Begin Motion
AM A	After Complete
MG "AT HOME"	Send Message
EN	End
#EDGE	Label
AC 2000000	Acceleration rate
DC 2000000	Deceleration rate
SP 8000	Speed
FE B	Find edge command
BG B	Begin motion
AM B	After complete
MG "FOUND HOME"	Send message
DP,0	Define position as 0
EN	End

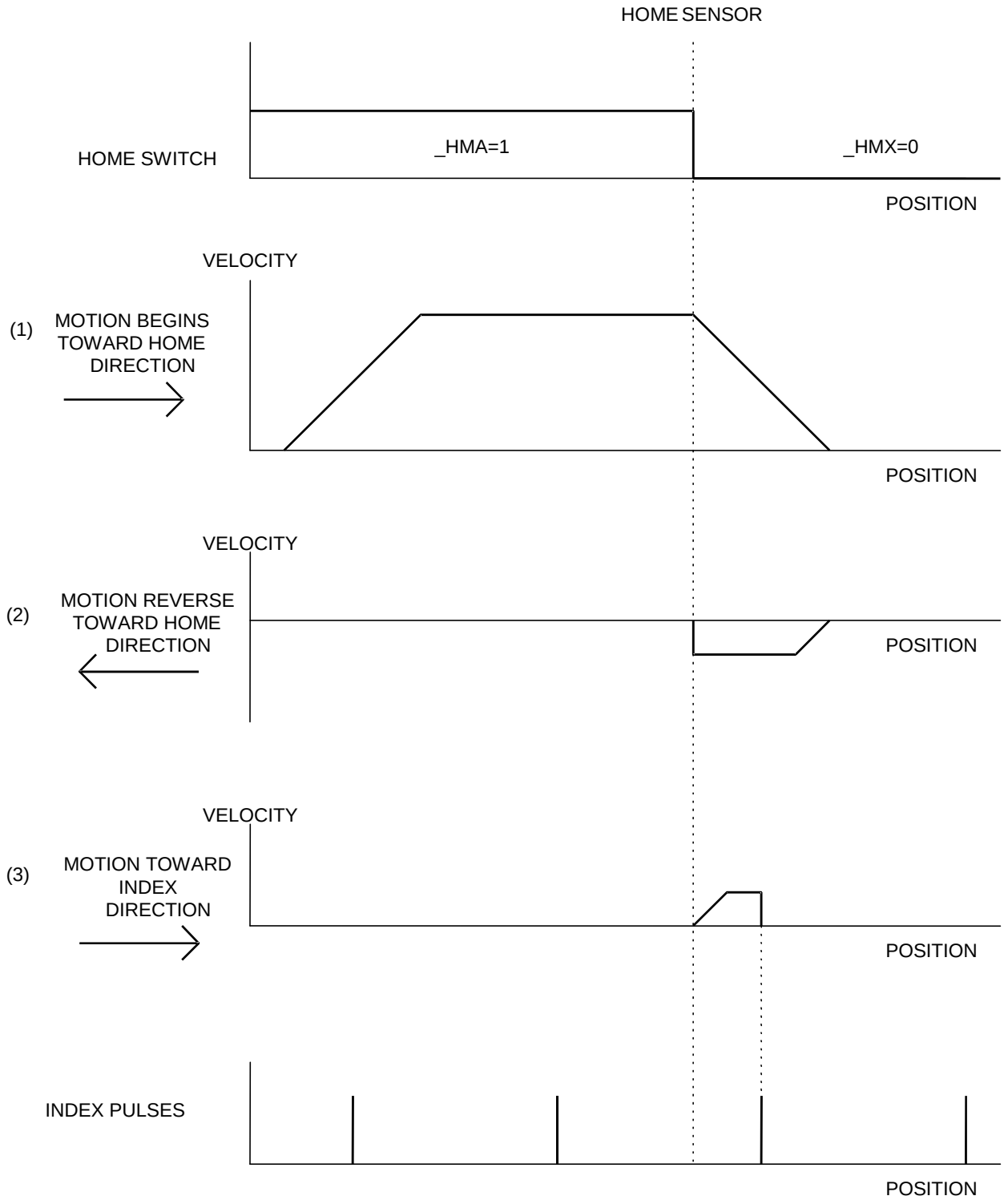


Figure 6.10 - Motion intervals in the Home sequence

Command Summary - Homing Operation

command	Description
FE ABCD	Find Edge Routine. This routine monitors the Home Input
FI ABCD	Find Index Routine - This routine monitors the Index Input
HM ABCD	Home Routine - This routine combines FE and FI as Described Above
SC ABCD	Stop Code
TS ABCD	Tell Status of Switches and Inputs

Operand Summary - Homing Operation

Operand	Description
_HMa	Contains the value of the state of the Home Input
_SCa	Contains stop code
_TSa	Contains status of switches and inputs

High Speed Position Capture (The Latch Function)

Often it is desirable to capture the position precisely for registration applications. The CDS-3310 provides a position latch feature. This feature allows the position of the main or auxiliary encoder to be captured when the latch input changes state. This function can be set up such that the position is captured when the latch input goes high or low. When the latch function is enabled for active low operation, the position will be captured within 12 microseconds. When the latch function is enabled for active high operation, the position will be captured within 35 microseconds. General input 1 is associated to the axis for position capture.

The CDS-3310 software commands, AL and RL, are used to arm the latch and report the latched position. The steps to use the latch are as follows:

1. Give the AL ABCD command to arm the latch for the main encoder and ALSASBSCSD for the auxiliary encoders.
2. Test to see if the latch has occurred (Input goes low) by using the _AL A or B or C or D command. Example, V1=_ALA returns the state of the A latch into V1. V1 is 1 if the latch has not occurred.
3. After the latch has occurred, read the captured position with the RL ABCD command or _RL ABCD.

NOTE: The latch must be re-armed after each latching event.

Example

```
#LATCH                  Latch program
JG,5000                 Jog B
BG B                    Begin motion on B axis
AL B                    Arm Latch for B axis
#WAIT                   #Wait label for loop
JP #WAIT,_ALB=1         Jump to #Wait label if latch has not occurred
Result=_RLB             Set 'Result' equal to the reported position of y axis
Result=                 Print result
EN                      End
```

Chapter 7 Application Programming

Overview

The CDS-3310 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the CDS-3310 memory freeing the host computer for other tasks. However, the host computer can send commands to the controller at any time, even while a program is being executed. Only ASCII commands can be used for application programming.

In addition to standard motion commands, the CDS-3310 provides commands that allow the CDS-3310 to make its own decisions. These commands include conditional jumps, event triggers and subroutines. For example, the command JP#LOOP, n<10 causes a jump to the label #LOOP if the variable n is less than 10.

For greater programming flexibility, the CDS-3310 provides user-defined variables, arrays and arithmetic functions. For example, with a cut-to-length operation, the length can be specified as a variable in a program which the operator can change as necessary.

The following sections in this chapter discuss all aspects of creating applications programs. The program memory size is 80 characters x 1000 lines.

Using the DOS Editor to Enter Programs

The CDS-3310 has an internal editor which may be used to create and edit programs in the controller's memory. The internal editor is opened by the command ED. Note that the command ED will not open the internal editor if issued from Galil's Window based software - in this case, a Windows based editor will be automatically opened. The Windows based editor provides much more functionality and ease-of-use, therefore, the internal editor is most useful when using a simple terminal with the controller and a Windows based editor is not available.

Once the ED command has been given, each program line is automatically numbered sequentially starting with 000. If no parameter follows the ED command, the editor prompter will default to the last line of the last program in memory. If desired, the user can edit a specific line number or label by specifying a line number or label following ED.

NOTE: ED command only accepts a parameter (such as #BEGIN) in DOS Window. For general purposes, the editing features in this section are not applicable when not in DOS mode.

Instruction	Interpretation
:ED	Puts Editor at end of last program

:ED 5	Puts Editor at line 5
:ED #BEGIN	Puts Editor at label #BEGIN

Line numbers appear as 000,001,002 and so on. Program commands are entered following the line numbers. Multiple commands may be given on a single line as long as the total number of characters doesn't exceed 80 characters per line.

While in the Edit Mode, the programmer has access to special instructions for saving, inserting and deleting program lines. These special instructions are listed below:

Edit Mode Commands

<RETURN>

Typing the return key causes the current line of entered instructions to be saved. The editor will automatically advance to the next line. Thus, hitting a series of <RETURN> will cause the editor to advance a series of lines. Note, changes on a program line will not be saved unless a <return> is given.

<cntrl>P

The <cntrl>P command moves the editor to the previous line.

<cntrl>I

The <cntrl>I command inserts a line above the current line. For example, if the editor is at line number 2 and <cntrl>I is applied, a new line will be inserted between lines 1 and 2. This new line will be labelled line 2. The old line number 2 is renumbered as line 3.

<cntrl>D

The <cntrl>D command deletes the line currently being edited. For example, if the editor is at line number 2 and <cntrl>D is applied, line 2 will be deleted. The previous line number 3 is now renumbered as line number 2.

<cntrl>Q

The <cntrl>Q quits the editor mode. In response, the CDS-3310 will return a colon.

After the Edit session is over, the user may list the entered program using the LS command. If no operand follows the LS command, the entire program will be listed. The user can start listing at a specific line or label using the operand n. A command and new line number or label following the start listing operand specifies the location at which listing is to stop.

Example

Instruction	Interpretation
:LS	List entire program
:LS 5	Begin listing at line 5
:LS 5,9	List lines 5 thru 9
:LS #A,9	List line label #A thru line 9
:LS #A, #A +5	List line label #A and additional 5 lines

Program Format

A DMC program consists of CDS-3310 instructions combined to solve a machine control application. Action instructions, such as starting and stopping motion, are combined with Program Flow instructions to form the complete program. Program flow instructions evaluate real-time conditions, such as elapsed time or motion complete, and alter program flow accordingly.

Each CDS-3310 instruction in a program must be separated by a delimiter. Valid delimiters are the semicolon (;) or carriage return. The semicolon is used to separate multiple instructions on a single program line where the maximum number of instructions on a line is limited by 80 characters. A carriage return enters the final command on a program line.

Using Labels in Programs

All CDS-3310 programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted.

Valid labels

#BEGIN

#SQUARE

#X1

#BEGIN1

Invalid labels

#1Square

#123

Example

Instruction	Interpretation
#START	Beginning of the Program
PR 10000,20000	Specify relative distances on A and B axes
BG AB	Begin Motion
AM	Wait for motion complete
WT 2000	Wait 2 sec
JP #START	Jump to label START
EN	End of Program

The above program moves A and B 10000 and 20000 units. After the motion is complete, the motors rest for 2 seconds. The cycle repeats indefinitely until the stop command is issued.

Special Labels

The CDS-3310 has some special labels, which are used to define input interrupt subroutines, limit switch subroutines, error handling subroutines, and command error subroutines. See the section on Auto-Start Routine.

The CDS-3310 has a special label for automatic program execution. A program which has been saved into the controller's non-volatile memory can be automatically executed upon power up or reset by

beginning the program with the label #AUTO. The program must be saved into non-volatile memory using the command, BP.

Automatic Subroutines for Monitoring Conditions:

#ININT	Label for Input Interrupt subroutine
#LIMSWI	Label for Limit Switch subroutine
#POSERR	Label for excess Position Error subroutine
#MCTIME	Label for timeout on Motion Complete trip point
#CMDERR	Label for incorrect command subroutine
#COMINT	Label for communication interrupt
#TCPERR	Label for TCP/IP communication error
#AMPERR	Label for Amplifier Error
#AUTOERR	Label for checksum error on power up. See _RS.

Commenting Programs

There are two methods for commenting programs. The first method uses the NO command and allows for comments to be embedded into Galil programs. The second method used the REM statement and requires the use of Galil software.

NO Command

The CDS-3310 provides a command, NO, for commenting programs. This command allows the user to include up to 78 characters on a single line after the NO command and can be used to include comments from the programmer as in the following example:

Instruction	Interpretation
#PATH	Label
NO This is a comment	Comment - No Operation

NOTE: The NO command is an actual controller command. Therefore, inclusion of the NO commands will require process time by the controller.

HINT: *Some users annotate their programs using the word "NOTE" since everything after the "NO" is a comment. Likewise an apostrophe (') can be used instead of the NO command when commenting programs.*

REM Command

If you are using Galil software to communicate with the CDS-3310 controller, you may also include REM statements. 'REM' statements begin with the word 'REM' and may be followed by any comments which are on the same line. The Galil terminal software will remove these statements when the program is downloaded to the controller. For example:

```
#PATH
REM 2-D CIRCULAR PATH
```

Executing Programs - Multitasking

The CDS-3310 can run up to 8 independent programs simultaneously. These programs are called threads and are numbered 0 through 7, where 0 is the main thread. Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion.

The main thread differs from the others in the following ways:

1. Only the main thread, thread 0, may use the input command, IN.

2. When automatic subroutines are implemented for limit switches, position errors or command errors, they are executed in thread 0.

To begin execution of the various programs, use the following instruction:

XQ #A, n

Where n indicates the thread number. To halt the execution of any thread, use the instruction

HX n

where n is the thread number.

Note that both the XQ and HX commands can be performed by an executing program.

The example below produces a waveform on Output 1 independent of a move.

Instruction	Interpretation
#TASK1	Task1 label
AT0	Initialize reference time
CB1	Clear Output 1
#LOOP1	Loop1 label
AT 10	Wait 10 msec from reference time
SB1	Set Output 1
AT -40	Wait 40 msec from reference, then initialize reference
CB1	Clear Output 1
JP #LOOP1	Repeat Loop1
#TASK2	Task2 label
XQ #TASK1,1	Execute Task1
#LOOP2	Loop2 label
PR 1000	Define relative distance
BGX	Begin motion
AMX	After motion done
WT 10	Wait 10 msec
JP #LOOP2,@IN[2]=1	Repeat motion unless Input 2 is low
HX	Halt all tasks

The program above is executed with the instruction XQ #TASK2,0 which designates TASK2 as the main thread (ie. Thread 0). #TASK1 is executed within TASK2.

Debugging Programs

The CDS-3310 provides commands and operands which are useful in debugging application programs. These commands include interrogation commands to monitor program execution, determine the state of the controller and the contents of the controllers program, array, and variable space. Operands also contain important status information which can help to debug a program.

Trace Command

The trace command causes the controller to send each line in a program to the host computer immediately prior to execution. Tracing is enabled with the command, TR1. TR0 turns the trace function off.

NOTE: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

Error Code Command

When there is a program error, the CDS-3310 halts the program execution at the point where the error occurs. To display the last line number of program execution, issue the command, MG _ED.

The user can obtain information about the type of error condition that occurred by using the command, TC1. This command reports back a number and a text message which describes the error condition. The command, TC0 or TC, will return the error code without the text message. For more information about the command, TC, see the Command Reference.

Stop Code Command

The status of motion for each axis can be determined by using the stop code command, SC. This can be useful when motion on an axis has stopped unexpectedly. The command SC will return a number representing the motion status. See the command reference for further information.

RAM Memory Interrogation Commands

For debugging the status of the program memory, array memory, or variable memory, the CDS-3310 has several useful commands. The command, DM ?, will return the number of array elements currently available. The command, DA ?, will return the number of arrays which can be currently defined. For example, a standard CDS-3310 will have a maximum of 8000 array elements in up to 30 arrays. If an array of 100 elements is defined, the command DM ? will return the value 7900 and the command DA ? will return 29.

To list the contents of the variable space, use the interrogation command LV (List Variables). To list the contents of array space, use the interrogation command, LA (List Arrays). To list the contents of the Program space, use the interrogation command, LS (List). To list the application program labels only, use the interrogation command, LL (List Labels).

Operands

In general, all operands provide information which may be useful in debugging an application program. Below is a list of operands which are particularly valuable for program debugging. To display the value of an operand, the message command may be used. For example, since the operand, _ED contains the last line of program execution, the command MG _ED will display this line number.

_ED contains the last line of program execution. Useful to determine where program stopped.

_DL contains the number of available labels.

_UL contains the number of available variables.

_DA contains the number of available arrays.

_DM contains the number of available array elements.

_AB contains the state of the Abort Input

_LFa contains the state of the forward limit switch for the 'a' axis

_LRa contains the state of the reverse limit switch for the 'a' axis

_RS contains the state of the power-up error status

Example

The following program has an error. It attempts to specify a relative movement while the A-axis is already in motion. When the program is executed, the controller stops at line 003. The user can then query the controller using the command, TC1. The controller responds with the corresponding explanation:

Instruction	Interpretation
:ED	Edit Mode
000 #A	Program Label
001 PR1000	Position Relative 1000
002 BGA	Begin
003 PR5000	Position Relative 5000
004 EN	End
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A
?003 PR5000	Error on Line 3
:TC1	Tell Error Code
?7 Command not valid while running.	Command not valid while running
:ED 3	Edit Line 3
003 AMX;PR5000;BGA	Add After Motion Done
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A

Program Flow Commands

The CDS-3310 provides instructions to control program flow. The CDS-3310 program sequencer normally executes program instructions sequentially. The program flow can be altered with the use of event triggers, trippoints, and conditional jump statements.

Event Triggers & Trippoints

To function independently from the host computer, the CDS-3310 can be programmed to make decisions based on the occurrence of an event. Such events include waiting for motion to be complete, waiting for a specified amount of time to elapse, or waiting for an input to change logic levels.

The CDS-3310 provides several event triggers that cause the program sequencer to halt until the specified event occurs. Normally, a program is automatically executed sequentially one line at a time. When an event trigger instruction is decoded, however, the actual program sequence is halted. The program sequence does not continue until the event trigger is "tripped". For example, the motion complete trigger can be used to separate two move sequences in a program. The commands for the second move sequence will not be executed until the motion is complete on the first motion sequence. In this way, the CDS-3310 can make decisions based on its own status or external events without intervention from a host computer.

CDS-3310 Event Triggers

Command	Function
AM A B C D E FG H or S	Halts program execution until motion is complete on the specified axes or motion sequence(s). AM with no parameter tests for motion complete on all axes. This command is useful for separating motion sequences in a program.
AD A or B or C or D or E or F or G or H	Halts program execution until position command has reached the specified relative distance from the start of the move. Only one axis may be specified at a time.
AR A or B or C or D or E or F or G or H	Halts program execution until after specified distance from the last AR or AD command has elapsed. Only one axis may be specified at a time.
AP A or B or C or D or E or F or G or H	Halts program execution until after absolute position occurs. Only one axis may be specified at a time.
MF A or B or C or D or E or F or G or H	Halt program execution until after forward motion reached absolute position. Only one axis may be specified. If position is already past the point, then MF will trip immediately. Will function on geared axis or aux. inputs.
MR A or B or C or D or E or F or G or H	Halt program execution until after reverse motion reached absolute position. Only one axis may be specified. If position is already past the point, then MR will trip immediately. Will function on geared axis or aux. inputs.
MC A or B or C or D or E or F or G or H	Halt program execution until after the motion profile has been completed and the encoder has entered or passed the specified position. TW A,B,C,D sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stopcode will be set to 99. An application program will jump to label #MCTIME.
AI +/- n	Halts program execution until after specified input is at specified logic level. n specifies input line. Positive is high logic level, negative is low level. n=1 through 8. n=17 through 56 with DB-28040.
AS A B C D E F G H	Halts program execution until specified axis has reached its slew speed.
AT +/-n	Halts program execution until n msec from reference time. AT 0 sets reference. AT n waits n msec from reference. AT -n waits n msec from reference and sets new reference after elapsed time.
WT n	Halts program execution until specified time in msec has elapsed.

Example- Multiple Move Sequence

The AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

Instruction	Interpretation
#TWOMOVE	Label
PR 2000	Position Command
BGA	Begin Motion
AMA	Wait for Motion Complete
PR 4000	Next Position Move
BGA	Begin 2nd move
EN	End program

Example- Set Output after Distance

Set output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

Instruction	Interpretation
#SETBIT	Label
SP 10000	Speed is 10000
PA 20000	Specify Absolute position
BGA	Begin motion
AD 1000	Wait until 1000 counts
SB1	Set output bit 1
EN	End program

Example- Repetitive Position Trigger

To set the output bit every 10000 counts during a move, the AR trippoint is used as shown in the next example.

Instruction	Interpretation
#TRIP	Label
JG 50000	Specify Jog Speed
BGA;n=0	Begin Motion
#REPEAT	# Repeat Loop
AR 10000	Wait 10000 counts
TPA	Tell Position
SB1	Set output 1
WT50	Wait 50 msec
CB1	Clear output 1
n=n+1	Increment counter
JP #REPEAT,n<5	Repeat 5 times
STA	Stop
EN	End

Example- Start Motion on Input

This example waits for input 1 to go low and then starts motion.

NOTE: The AI command actually halts execution of the program until the input occurs. If you do not want to halt the program sequences, you can use the Input Interrupt function (II) or use a conditional jump on an input, such as JP #GO,@IN[1] = -1.

Instruction	Interpretation
#INPUT	Program Label
AI-1	Wait for input 1 low
PR 10000	Position command
BGA	Begin motion
EN	End program

Example - Set Output when At Speed

Instruction	Interpretation
#ATSPEED	Program Label
JG 50000	Specify jog speed
AC 10000	Acceleration rate
BGA	Begin motion
ASA	Wait for at slew speed 50000
SB1	Set output 1
EN	End program

Example - Multiple Move with Wait

This example makes multiple relative distance moves by waiting for each to be complete before executing new moves.

Instruction	Interpretation
#MOVES	Label
PR 12000	Distance
SP 20000	Speed
AC 100000	Acceleration
BGA	Start Motion
AD 10000	Wait a distance of 10,000 counts
SP 5000	New Speed
AMA	Wait until motion is completed
WT 200	Wait 200 ms
PR -10000	New Position
SP 30000	New Speed
AC 150000	New Acceleration
BGA	Start Motion
EN	End

Example- Define Output Waveform Using AT

The following program causes Output 1 to be high for 10 msec and low for 40 msec. The cycle repeats every 50 msec.

Instruction	Interpretation
#OUTPUT	Program label
AT0	Initialize time reference
SB1	Set Output 1
#LOOP	Loop
AT 10	After 10 msec from reference,
CB1	Clear Output 1
AT -40	Wait 40 msec from reference and reset reference
SB1	Set Output 1
JP #LOOP	Loop
EN	

Conditional Jumps

The CDS-3310 provides Conditional Jump (JP) and Conditional Jump to Subroutine (JS) instructions for branching to a new program location based on a specified condition. The conditional jump determines if a condition is satisfied and then branches to a new location or subroutine. Unlike event triggers, the conditional jump instruction does not halt the program sequence. Conditional jumps are useful for testing events in real-time. They allow the CDS-3310 to make decisions without a host computer. For example, the CDS-3310 can decide between two motion profiles based on the state of an input line.

Command Format - JP and JS

FORMAT:	DESCRIPTION
JS destination, logical condition	Jump to subroutine if logical condition is satisfied
JP destination, logical condition	Jump to location if logical condition is satisfied

The destination is a program line number or label where the program sequencer will jump if the specified condition is satisfied. Note that the line number of the first line of program memory is 0. The comma designates "IF". The logical condition tests two operands with logical operators.

Logical operators:

OPERATOR	DESCRIPTION
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal

Conditional Statements

The conditional statement is satisfied if it evaluates to any value other than zero. The conditional statement can be any valid CDS-3310 numeric operand, including variables, array elements, numeric

values, functions, keywords, and arithmetic expressions. If no conditional statement is given, the jump will always occur.

Number	V1=6
Numeric Expression	V1=V7*6
	@ABS[V1]>10
Array Element	V1<Count[2]
Variable	V1<V2
Internal Variable	_TPA=0
	_TVA>500
I/O	V1>_TTX
	@IN[1]=0

Multiple Conditional Statements

The CDS-3310 will accept multiple conditions in a single jump statement. The conditional statements are combined in pairs using the operands "&" and "|". The "&" operand between any two conditions, requires that both statements must be true for the combined statement to be true. The "|" operand between any two conditions, requires that only one statement be true for the combined statement to be true.

NOTE: Each condition must be placed in parenthesis for proper evaluation by the controller. In addition, the CDS-3310 executes operations from left to right. For further information on Mathematical Expressions and the bit-wise operators '&' and '|', see pg 35.

For example, using variables named V1, V2, V3 and V4:

JP #TEST, (V1<V2) & (V3<V4)

In this example, this statement will cause the program to jump to the label #TEST if V1 is less than V2 and V3 is less than V4. To illustrate this further, consider this same example with an additional condition:

JP #TEST, ((V1<V2) & (V3<V4)) | (V5<V6)

This statement will cause the program to jump to the label #TEST under two conditions; 1. If V1 is less than V2 and V3 is less than V4. OR 2. If V5 is less than V6.

Examples

If the condition for the JP command is satisfied, the controller branches to the specified label or line number and continues executing commands from this point. If the condition is not satisfied, the controller continues to execute the next commands in sequence.

Instruction	**Interpretation**
JP #LOOP,count<10	Jump to #Loop if the variable, COUNT, is less than 10
JS #MOVE2,@IN[1]=1	Jump to subroutine #MOVE2 if input 1 is logic level high. After the subroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called.
JP #BLUE,@ABS[v2]>2	Jump to #BLUE if the absolute value of variable, v2, is greater than 2
JP #C,v1*v7<=v8*v2	Jump to #C if the value of v1 times v7 is less than or equal to the value of V8*v2
JP#A	Jump to #A

Move the A motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

Instruction	Interpretation
#BEGIN	Begin Program
count=10	Initialize loop counter
#LOOP	Begin loop
PA 1000	Position absolute 1000
BGA	Begin move
AMA	Wait for motion complete
WT 100	Wait 100 msec
PA 0	Position absolute 0
BGA	Begin move
AMA	Wait for motion complete
WT 100	Wait 100 msec
count = count -1	Decrement loop counter
JP #LOOP, count >0	Test for 10 times thru loop
EN	End Program

IF, ELSE, AND ENDIF

The CDS-3310 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

Using the IF and ENDIF Commands

An IF conditional statement is formed by the combination of an IF and ENDIF command. The IF command has as it's arguments one or more conditional statements. If the conditional statement(s) evaluates true, the command interpreter will continue executing commands which follow the IF command. If the conditional statement evaluates false, the controller will ignore commands until the associated ENDIF command is executed OR an ELSE command occurs in the program (see discussion of ELSE command below).

NOTE: An ENDIF command must always be executed for every IF command that has been executed. It is recommended that the user not include jump commands inside IF conditional statements since this causes re-direction of command execution. In this case, the command interpreter may not execute an ENDIF command.

Using the ELSE Command

The ELSE command is an optional part of an IF conditional statement and allows for the execution of command only when the argument of the IF command evaluates False. The ELSE command must occur after an IF command and has no arguments. If the argument of the IF command evaluates false, the controller will skip commands until the ELSE command. If the argument for the IF command evaluates true, the controller will execute the commands between the IF and ELSE command.

Nesting IF Conditional Statements

The CDS-3310 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the CDS-3310 allows up to 255 IF conditional statements to be nested. This is a very powerful technique allowing the user to specify a variety of different cases for branching.

Command Format - IF, ELSE and ENDIF

Format:	description
IF conditional statement(s)	Execute commands proceeding IF command (up to ELSE command) if conditional statement(s) is true, otherwise continue executing at ENDIF command or optional ELSE command.
ELSE	Optional command. Allows for commands to be executed when argument of IF command evaluates not true. Can only be used with IF command.
ENDIF	Command to end IF conditional statement. Program must have an ENDIF command for every IF command.

Instruction	Interpretation
#TEST	Begin Main Program "TEST"
II,,3	Enable interrupts on input 1 and input 2
MG "WAITING FOR INPUT 1, INPUT 2"	Output message
#LOOP	Label to be used for endless loop
JP #LOOP	Endless loop
EN	End of main program
#ININT	Input Interrupt Subroutine
IF (@IN[1]=0)	IF conditional statement based on input 1
IF (@IN[2]=0)	2nd IF executed if 1st IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"	Message executed if 2nd IF is true
ELSE	ELSE command for 2nd IF statement
MG "ONLY INPUT 1 IS ACTIVE	Message executed if 2nd IF is false
ENDIF	End of 2nd conditional statement
ELSE	ELSE command for 1st IF statement
MG"ONLY INPUT 2 IS ACTIVE"	Message executed if 1st IF statement
ENDIF	End of 1st conditional statement
#WAIT	Label to be used for a loop
JP#WAIT,(@IN[1]=0) | (@IN[2]=0)	Loop until Input 1& 2 are not active
RI0	End Input Interrupt Routine without restoring trippoints

Subroutines

A subroutine is a group of instructions beginning with a label and ending with an end command (EN). Subroutines are called from the main program with the jump subroutine instruction JS, followed by a label or line number, and conditional statement. Up to 8 subroutines can be nested. After the subroutine is executed, the program sequencer returns to the program location where the subroutine was called unless the subroutine stack is manipulated as described in the following section.

An example of a subroutine is given below.

Instruction	Interpretation
#M	Begin Main Program
CB1	Clear Output Bit 1
JS #MySub	Jump to square subroutine
EN	End Main Program
#MySub	subroutine
MG “In subroutine”	
EN	End subroutine

Stack Manipulation

It is possible to manipulate the subroutine stack by using the ZS command. Every time a JS instruction, interrupt or automatic routine (such as #POSERR or #LIMSWI) is executed, the subroutine stack is incremented by 1. Normally the stack is restored with an EN instruction. Occasionally it is desirable not to return back to the program line where the subroutine or interrupt was called. The ZS1 command clears 1 level of the stack. This allows the program sequencer to continue to the next line. The ZS0 command resets the stack to its initial value. For example, if a limit occurs and the #LIMSWI routine is executed, it is often desirable to restart the program sequence instead of returning to the location where the limit occurred. To do this, give a ZS command at the end of the #LIMSWI routine.

Auto-Start Routine

The CDS-3310 has a special label for automatic program execution. A program which has been saved into the controllers non-volatile memory can be automatically executed upon power up or reset by beginning the program with the label #AUTO. The program must be saved into non-volatile memory using the command, BP.

Automatic Subroutines for Monitoring Conditions

Often it is desirable to monitor certain conditions continuously without tying up the host or CDS-3310 program sequences. The CDS-3310 can monitor several important conditions in the background. These conditions include checking for the occurrence of a limit switch, a defined input, position error, or a command error. Automatic monitoring is enabled by inserting a special, predefined label in the applications program. The pre-defined labels are:

SUBROUTINE	DESCRIPTION
#LIMSWI	Limit switch on any axis goes low
#ININT	Input specified by II goes low
#POSERR	Position error exceeds limit specified by ER
#MCTIME	Motion complete timeout occurred. Timeout period set by TW command
#COMINT	Communication interrupt
#CMDERR	Bad command given
#TCPERR	TCP/IP communication error
#AMPERR	Amplifier error
#AUTOERR	Label for checksum error on power up. See _RS.

For example, the #POSERR subroutine will automatically be executed when any axis exceeds its position error limit. The commands in the #POSERR subroutine could decode which axis is in error and take the appropriate action. In another example, the #ININT label could be used to designate an

input interrupt subroutine. When the specified input occurs, the program will be executed automatically.

NOTE: An application program must be running for automatic monitoring to function.

Example - Limit Switch:

This program prints a message upon the occurrence of a limit switch. Note, for the #LIMSWI routine to function, the CDS-3310 must be executing an applications program from memory. This can be a very simple program that does nothing but loop on a statement, such as #LOOP;JP #LOOP;EN. Motion commands, such as JG 5000 can still be sent from the PC even while the "dummy" applications program is being executed.

Instruction	Interpretation
:ED	Edit Mode
000 #LOOP	Dummy Program
001 JP #LOOP;EN	Jump to Loop
002 #LIMSWI	Limit Switch Label
003 MG "LIMIT OCCURRED"	Print Message
004 RE	Return to main program
<control> Q	Quit Edit Mode
:XQ #LOOP	Execute Dummy Program
:JG 5000	Jog
:BGA	Begin Motion

Now, when a forward limit switch occurs on the A axis, the #LIMSWI subroutine will be executed

Notes regarding the #LIMSWI Routine:

1) The RE command is used to return from the #LIMSWI subroutine.

2) The #LIMSWI subroutine will be re-executed if the limit switch remains active.

The #LIMSWI routine is only executed when the motor is being commanded to move.

Example - Position Error

Instruction	Interpretation
:ED	Edit Mode
000 #LOOP	Dummy Program
001 JP #LOOP;EN	Loop
002 #POSERR	Position Error Routine
003 v1=_TEA	Read Position Error
004 MG "EXCESS POSITION ERROR"	Print Message
005 MG "ERROR=",v1=	Print Error
006 RE	Return from Error
<control> Q	Quit Edit Mode
:XQ #LOOP	Execute Dummy Program
:JG 100000	Jog at High Speed
:BGX	Begin Motion

Now, when position error exceeds error limit, ER, on the A axis, the #POSERR subroutine will be executed. The RE command is used to return from the #POSERR subroutine.

Example - Input Interrupt

Instruction	Interpretation
#A	Label
II1	Input Interrupt on 1
JG 30000,,,60000	Jog
BGAD	Begin Motion
#LOOP;JP#LOOP;EN	Loop
#ININT	Input Interrupt
STAD;AM	Stop Motion
#TEST;JP #TEST, @IN[1]=0	Test for Input 1 still low
JG 30000,,,6000	Restore Velocities
BGAD	Begin motion
RI0	Return from interrupt routine to Main Program and do not re-enable trippoints

Example - Motion Complete Timeout

Instruction	Interpretation
#BEGIN	Begin main program
TW 1000	Set the time out to 1000 ms
PA 10000	Position Absolute command
BGA	Begin motion
MCA	Motion Complete trip point
EN	End main program
#MCTIME	Motion Complete Subroutine
MG “A fell short”	Send out a message
EN	End subroutine

This simple program will issue the message “A fell short” if the A axis does not reach the commanded position within 1 second of the end of the profiled move.

Example - Communication Interrupt

A CDS-3310 is used to move the A axis back and forth from 0 to 10000. This motion can be paused, resumed and stopped via input from an auxiliary port terminal.

Instruction	Interpretation
#BEGIN	Label for beginning of program
CI 2,1	Set up communication interrupt for serial port on CDS-3310
MG "Type 0 to stop motion"	Message out of serial port
MG "Type 1 to pause motion"	Message out of serial port
MG "Type 2 to resume motion"	Message out of serial port
rate=2000	Variable to remember speed
SPA=rate	Set speed of A axis motion
#LOOP	Label for Loop
PAA=10000	Move to absolute position 10000
BGA	Begin Motion on A axis

AMA	Wait for motion to be complete
PAA=0	Move to absolute position 0
BGA	Begin Motion on A axis
AMA	Wait for motion to be complete
JP #LOOP	Continually loop to make back and forth motion
EN	End main program
#COMINT	Interrupt Routine
#STOP,P1CH="0"	Check for S (stop motion)
JP #PAUSE,P1CH="1"	Check for P (pause motion)
JP #RESUME,P1CH="2"	Check for R (resume motion)
EN1,1	Do nothing
#STOP	Routine for stopping motion
STA;ZS;EN	Stop motion on A axis; Zero program stack; End Program
#PAUSE	Routine for pausing motion
rate=_SPA	Save current speed setting of A axis motion
SPA=0	Set speed of A axis to zero (allows for pause)
EN1,1	Re-enable trip-point and communication interrupt
#RESUME	Routine for resuming motion
SPA=rate	Set speed on A axis to original speed
EN1,1	Re-enable trip-point and communication interrupt

Example - Command Error

Instruction	Interpretation
#BEGIN	Begin main program
IN "ENTER SPEED", speed	Prompt for speed
JG speed;BGA;	Begin motion
JP #BEGIN	Repeat
EN	End main program
#CMDERR	Command error utility
JP#DONE,_ED<>2	Check if error on line 2
JP#DONE,_TC<>6	Check if out of range
MG "SPEED TOO HIGH"	Send message
MG "TRY AGAIN"	Send message
ZS1	Adjust stack
JP #BEGIN	Return to main program
#DONE	End program if other error
ZS0	Zero stack
EN	End program

The above program prompts the operator to enter a jog speed. If the operator enters a number out of range (greater than 8 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

In multitasking applications, there is an alternate method for handling command errors from different threads. Using the XQ command along with the special operands described below allows the controller to either skip or retry invalid commands.

OPERAND	FUNCTION
_ED1	Returns the number of the thread that generated an error
_ED2	Retry failed command (operand contains the location of the failed command)
_ED3	Skip failed command (operand contains the location of the command after the failed command)

The operands are used with the XQ command in the following format:

XQ _ED2 (or _ED3),_ED1,1

Where the ",1" at the end of the command line indicates a restart; therefore, the existing program stack will not be removed when the above format executes.

The following example shows an error correction routine which uses the operands.

Example - Command Error with Multitasking

Instruction	Interpretation
#A	Begin thread 0 (continuous loop)
JP#A	
EN	End of thread 0
#B	Begin thread 1
n=-1	Create new variable
KP n	Set KP to value of n, an invalid value
TY	Issue invalid command
EN	End of thread 1
#CMDERR	Begin command error subroutine
IF (_TC=6)	If error is out of range (KP -1)
n=1	Set n to a valid number
XQ _ED2,_ED1,1	Retry KP n command
ENDIF	
IF (_TC=1)	If error is invalid command (TY)
XQ _ED3,_ED1,1	Skip invalid command
ENDIF	
EN	End of command error routine

Example – Ethernet Communication Error

This simple program executes in the CDS-3310 and indicates (via the serial port) when a communication handle fails. By monitoring the serial port, the user can re-establish communication if needed.

Instruction	Interpretation
#LOOP	Simple program loop
JP#LOOP	
EN	
#TCPERR	Ethernet communication error auto routine
MG {P1}_IA4	Send message to serial port indicating which handle did not receive proper acknowledgment.
RE	

Mathematical and Functional Expressions

Mathematical Operators

For manipulation of data, the CDS-3310 provides the use of the following mathematical operators:

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
&	Logical And (Bit-wise)
|	Logical Or (On some computers, a solid vertical line appears as a broken line)
()	Parenthesis

The numeric range for addition, subtraction and multiplication operations is +/-2,147,483,647.9999. The precision for division is 1/65,000.

Mathematical operations are executed from left to right. Calculations within a parentheses have precedence.

speed=7.5*V1/2	The variable 'speed' is equal to 7.5 multiplied by V1 and divided by 2
count= count +2	The variable 'count' is equal to the current value plus 2.
result=_TPA-(@COS[45]*40)	Puts the position of A - 28.28 in 'result' (40 * cosine of 45° is 28.28)
temp=@IN[1]&@IN[2]	'temp' is equal to 1 only if Input 1 and Input 2 are high

Bit-Wise Operators

The mathematical operators & and | are bit-wise operators. The operator, &, is a Logical And. The operator, |, is a Logical Or. These operators allow for bit-wise operations on any valid CDS-3310 numeric operand, including variables, array elements, numeric values, functions, keywords and arithmetic expressions. The bit-wise operators may also be used with strings. This is useful for separating characters from an input string. When using the input command for string input, the input variable will hold up to 6 characters. These characters are combined into a single value which is represented as 32 bits of integer and 16 bits of fraction. Each ASCII character is represented as one byte (8 bits), therefore the input variable can hold up to six characters. The first character of the string will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations as illustrated in the following example:

Instruction	Interpretation
#TEST	Begin main program
IN "ENTER",len{S6}	Input character string of up to 6 characters into variable 'len'
flen=@FRAC[len]	Define variable 'flen' as fractional part of variable 'len'
flen=$10000* flen	Shift flen by 32 bits (IE - convert fraction, flen, to integer)
len1=(flen &$00FF)	Mask top byte of flen and set this value to variable

```
                                        'len1'
len2=( flen &$FF00)/$100                Let variable, 'len2' = top byte of flen
len3= len &$000000FF                    Let variable, 'len3' = bottom byte of len
len4=( len &$0000FF00)/$100             Let variable, 'len4' = second byte of len
len5=( len &$00FF0000)/$10000           Let variable, 'len5' = third byte of len
len6=( len &$FF000000)/$1000000         Let variable, 'len6' = fourth byte of len
MG len6 {S4}                            Display 'len6' as string message of up to 4 chars
MG len5 {S4}                            Display 'len5' as string message of up to 4 chars
MG len4 {S4}                            Display 'len4' as string message of up to 4 chars
MG len3 {S4}                            Display 'len3' as string message of up to 4 chars
MG len2 {S4}                            Display 'len2' as string message of up to 4 chars
MG len1 {S4}                            Display 'len1' as string message of up to 4 chars
EN
```

This program will accept a string input of up to 6 characters, parse each character, and then display each character. Notice also that the values used for masking are represented in hexadecimal (as denoted by the preceding '$'). For more information, see section *Sending Message*s.

To illustrate further, if the user types in the string "TESTME" at the input prompt, the controller will respond with the following:

```
T                                       Response from command MG len6 {S4}
E                                       Response from command MG len5 {S4}
S                                       Response from command MG len4 {S4}
T                                       Response from command MG len3 {S4}
M                                       Response from command MG len2 {S4}
E                                       Response from command MG len1 {S4}
```

Functions

FUNCTION	DESCRIPTION
@SIN[n]	Sine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@COS[n]	Cosine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@TAN[n]	Tangent of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@ASIN*[n]	Arc Sine of n, between -90° and +90°. Angle resolution in 1/64000 degrees.
@ACOS* [n}	Arc Cosine of n, between 0 and 180°. Angle resolution in 1/64000 degrees.
@ATAN* [n]	Arc Tangent of n, between -90° and +90°. Angle resolution in 1/64000 degrees
@COM[n]	One's Complement of n
@ABS[n]	Absolute value of n
@FRAC[n]	Fraction portion of n
@INT[n]	Integer portion of n
@RND[n]	Round of n (Rounds up if the fractional part of n is .5 or greater)
@SQR[n]	Square root of n (Accuracy is +/-.0001)
@IN[n]	Return digital input at general input n (where n starts at 1)
@OUT[n]	Return digital output at general output n (where n starts at 1)

* Note that these functions are multi-valued. An application program may be used to find the correct band.

Functions may be combined with mathematical expressions. The order of execution of mathematical expressions is from left to right and can be over-ridden by using parentheses.

Instruction	Interpretation
v1=@ABS[v7]	The variable, V1, is equal to the absolute value of variable V7.
v2=5*@SIN[pos]	The variable, V2, is equal to five times the sine of the variable, POS.
v3=@IN[1]	The variable, V3, is equal to the digital value of input 1.

Variables

For applications that require a parameter that is variable, the CDS-3310 provides 510 variables. These variables can be numbers or strings. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by program calculations. For example, a cut-to-length application may require that a cut length be variable.

Instruction	Interpretation
PR posa	Assigns variable 'posa' to PR command
JG rpmb*70	Assigns variable 'rpmb' multiplied by 70 to JG command.

Programmable Variables

Each variable is defined by a name which can be up to eight characters. The name must start with an alphabetic character, however,

numbers are permitted in the rest of the name. Spaces are not permitted. Variables can be upper or lowercase, or any combination. Variables are case sensitive, SPEEDC≠speedC. Variable names should not be the same as CDS-3310 instructions. For example, PR is not a good choice for a variable name.

Examples of valid and invalid variable names are:

Valid Variable Names

POSA

pos1

speedC

Invalid Variable Names

```
REALLONGNAME    ; Cannot have more than 8 characters
123             ; Cannot begin variable name with a number
SPEED C         ; Cannot have spaces in the name
```

Assigning Values to Variables

Assigned values can be numbers, internal variables and keywords, functions, controller parameters and strings;

The range for numeric variable values is 4 bytes of integer (2^{31})followed by two bytes of fraction (+/-2,147,483,647.9999).

Numeric values can be assigned to programmable variables using the equal sign.

Any valid CDS-3310 function can be used to assign a value to a variable. For example, V1=@ABS[v2] or v2=@IN[1]. Arithmetic operations are also permitted.

To assign a string value, the string must be in quotations. String variables can contain up to six characters which must be in quotation.

Instruction	Interpretation
posA=_TPA	Assigns returned value from TPA command to variable posA
SPEED=5.75	Assigns value 5.75 to variable SPEED
input=@IN[2]	Assigns logical value of input 2 to variable input
v2=v1+v3*v4	Assigns the value of v1 plus v3 times v4 to the variable v2.
Var="CAT"	Assign the string, CAT, to Var

Assigning Variable Values to Controller Parameters

Variable values may be assigned to controller parameters such as KD or PR.

PR v1	Assign v1 to PR command
SP vS*2000	Assign vS*2000 to SP command

Displaying the value of variables at the terminal

Variables may be sent to the screen using the format, 'variable'=. For example, v1= , returns the value of the variable v1.

Operands

Operands allow motion or status parameters of the CDS-3310 to be incorporated into programmable variables and expressions. Most CDS-3310 commands have an equivalent operand which is designated by adding an underscore (_) prior to the CDS-3310 command. The command reference indicates which commands have an associated operand.

Status commands such as Tell Position return actual values, whereas action commands such as KP or SP return the values stored in the CDS-3310 registers. The axis designation is required following the command.

Instruction	Interpretation
posA=_TPA	Assigns value from Tell Position A to the variable posA.
JP #LOOP,_TEA>5	Jump to #LOOP if the position error of A is greater than 5
JP #ERROR,_TC=1	Jump to #ERROR if the error code equals 1.

Operands can be used in an expression and assigned to a programmable variable, but they cannot be assigned a value. For example: _TPA=2 is invalid.

Common Operands

Operand	Function
_BGn	*Returns a 1 if motion on axis 'n' is complete, otherwise returns 0.
_BN	*Returns serial # of the board.

_DA	*Returns the number of arrays available
_DL	*Returns the number of available labels for programming
_DM	*Returns the available array memory
_HMn	*Returns status of Home Switch of axis 'n' (equals 0 or 1)
_LFn	Returns status of Forward Limit switch input of axis 'n' (equals 0 or 1)
_LRn	Returns status of Reverse Limit switch input of axis 'n' (equals 0 or 1)
_UL	*Returns the number of available variables
TIME	Free-Running Real Time Clock (off by 2.4% - Resets with power-on). **NOTE:** TIME does not use an underscore character (_) as other keywords.

*These keywords have corresponding commands while the keywords _LF, _LR, and TIME do not have any associated commands. All keywords are listed in the Command Reference.

v1=_LFA	Assign v1 the state of the Forward Limit Switch on the A-axis
v3=TIME	Assign v3 the current value of the time clock
v4=_HMD	Assign v4 the logical state of the Home input on the D-axis

Arrays

For storing and collecting numerical data, the CDS-3310 provides array space for 8000 elements. The arrays are one dimensional and up to 30 different arrays may be defined. Each array element has a numeric range of 4 bytes of integer (2^{31})followed by two bytes of fraction (+/-2,147,483,647.9999).

Arrays can be used to capture real-time data, such as position and torque values. In the contouring mode, arrays are convenient for holding the points of a position trajectory in a record and playback application.

Defining Arrays

An array is defined with the command DM. The user must specify a name and the number of entries to be held in the array. An array name can contain up to eight characters, starting with an uppercase alphabetic character. The number of entries in the defined array is enclosed in [ ].

DM posA[7]	Defines an array names posA with seven entries
DM speed[100]	Defines an array named speed with 100 entries
DM posA[0]	Frees array space

Assignment of Array Entries

Like variables, each array element can be assigned a value. Assigned values can be numbers or returned values from instructions, functions and keywords.

Array elements are addressed starting at count 0. For example the first element in the posA array (defined with the DM command, DM posA[7]) would be specified as posA[0].

Values are assigned to array entries using the equal sign. Assignments are made one element at a time by specifying the element number with the associated array name.

NOTE: Arrays must be defined using the command, DM, before assigning entry values.

DM speed[10]	Dimension Speed Array
speed[1]=7650.2	Assigns the first element of the array the value 7650.2
speed[1]=	Returns array element value
posXA[10]=_TPA	Assigns the 10th element the position of A
con[2]=@COS[POS]*2	Assigns the 2nd element of the array the cosine of POS * 2.
timer[1]=TIME	Assigns the 1st element of the array 'timer'

Using a Variable to Address Array Elements

An array element number can also be a variable. This allows array entries to be assigned sequentially using a counter.

Instruction	Interpretation
#A	Begin Program
count=0;DM POS[10]	Initialize counter and define array
#LOOP	Begin loop
WT 10	Wait 10 msec
POS[count]=_TPA	Record position into array element
POS[count]=	Report position
count = count +1	Increment counter
JP #LOOP, count <10	Loop until 10 elements have been stored
EN	End Program

The above example records 10 position values at a rate of one value per 10 msec. The values are stored in an array named POS. The variable 'count' is used to increment the array element counter. The above example can also be executed with the automatic data capture feature described below.

Uploading and Downloading Arrays to On Board Memory

Arrays may be uploaded and downloaded using the QU and QD commands.

QU array[],start,end,delim

QD array[],start,end

where array is an array name such as A[].

Start is the first element of array (default=0)

End is the last element of array (default=last element)

Delim specifies whether the array data is seperated by a comma (delim=1) or a carriage return (delim=0).

The file is terminated using <control>Z, <control>Q, <control>D or \.

Automatic Data Capture into Arrays

The CDS-3310 provides a special feature for automatic capture of data such as position, position error, inputs or torque. This is useful for teaching motion trajectories or observing system performance. Up to eight types of data can be captured and stored in eight arrays. The capture rate or time interval may be specified. Recording can be done as a one time event or as a circular continuous recording.

Command Summary - Automatic Data Capture

command	description
RA n[],m[],o[],p[]	Selects up to four arrays for data capture. The arrays must be defined with the DM command.
RD type1,type2,type3,type4	Selects the type of data to be recorded, where type1, type2, type3, and type 4 represent the various types of data (see command reference). The order of data type is important and corresponds with the order of n,m,o,p arrays in the RA command.
RC n,m	The RC command begins data collection. Sets data capture time interval where n is an integer between 1 and 8 and designates 2^n msec between data. m is optional and specifies the number of elements to be captured. If m is not defined, the number of elements defaults to the smallest array defined by DM. When m is a negative number, the recording is done continuously in a circular manner. _RD is the recording pointer and indicates the address of the next array element. N=0 stops recording.
RC?	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress

Operand Summary - Automatic Data Capture

_RC	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress
_RD	Returns address of next array element.

Example - Recording into An Array

Instruction	Interpretation
#RECORD	Begin program
DM apos[300],bpos[300]	Define A,B position arrays
DM aerr[300],berr[300]	Define A,B error arrays
RA apos [],aerr[],bpos[],berr[]	Select arrays for capture
RD _TPA,_TEA,_TPB,_TEB	Select data types
PR 10000,20000	Specify move distance
RC1	Start recording now, at rate of 2 msec
BG AB	Begin motion
#A;JP #A,_RC=1	Loop until done
MG "DONE"	Print message
EN	End program
#PLAY	Play back
N=0	Initial Counter
JP# DONE,N>300	Exit if done
n=	Print Counter
apos [N]=	Print X position
bpos [N]=	Print Y position
aerr[N]=	Print X error
berr[N]=	Print Y error
n=n+1	Increment Counter
#DONE	Done
EN	End Program

Deallocating Array Space

Array space may be deallocated using the DA command followed by the array name. DA*[0] deallocates all the arrays.

Input of Data (Numeric and String—Serial Port Only)

Input of Data

The command, IN, is used to prompt the user to input numeric or string data. Using the IN command, the user may specify a message prompt by placing a message in quotations. When the controller executes an IN command, the controller will wait for the input of data. The input data is assigned to the specified variable or array element.

Example- Inputting Numeric Data

```
#A
IN "Enter Length",lenA
EN
```

In this example, the message “Enter Length” is displayed on the computer screen. The controller waits for the operator to enter a value. The operator enters the numeric value which is assigned to the variable, lenA. (**NOTE:** Do not include a space between the comma at the end of the input message and the variable name.)

Example- Cut-to-Length

In this example, a length of material is to be advanced a specified distance. When the motion is complete, a cutting head is activated to cut the material. The length is variable, and the operator is prompted to input it in inches. Motion starts with a start button which is connected to input 1.

The load is coupled with a 2 pitch lead screw. A 2000 count/rev encoder is on the motor, resulting in a resolution of 4000 counts/inch. The program below uses the variable len, to length. The IN command is used to prompt the operator to enter the length, and the entered value is assigned to the variable LEN.

Instruction	Interpretation
#BEGIN	LABEL
AC 800000	Acceleration
DC 800000	Deceleration
SP 5000	Speed
len=3.4	Initial length in inches
#CUT	Cut routine
AI1	Wait for start signal
IN "enter Length(IN)", len	Prompt operator for length in inches
PR len *4000	Specify position in counts
BGA	Begin motion to move material
AMA	Wait for motion done
SB1	Set output to cut
WT100;CB1	Wait 100 msec, then turn off cutter
JP #CUT	Repeat process

EN	End program

Operator Data Entry Mode

The Operator Data Entry Mode provides for un-buffered data entry through the RS-232 port. This mode is entered and exited by the CI command.

NOTE: Operator Data Entry Mode cannot be used for high rate data transfer.

To capture and decode characters in the Operator Data Mode, the controller provides the following keywords:

Keyword	Function
P1CH	Contains the last character received
P1ST	Contains the received string
P1NM	Contains the received number
P1CD	Contains the status code: -1 mode disabled 0 nothing received 1 received character, but not <enter> 2 received string, not a number 3 received number

NOTE: The value of P1CD returns to zero after the corresponding string or number is read.

These keywords may be used in an applications program to decode data and they may also be used in conditional statements with logical operators.

Example

Instruction	Interpretation
JP #LOOP,P1CD< >3	Checks to see if status code is 3 (number received)
JP #P,P1CH="V"	Checks if last character received was a V
PR P1NM	Assigns received number to position
JS #XAXIS,P1ST="X"	Checks to see if received string is X

Communication Interrupt

The controller provides a special interrupt for communication allowing the application program to be interrupted by input from the user. The interrupt is enabled using the CI command. The syntax for the command is CI n,m:

n = 0	Don't interrupt Port 1
n = 1	Interrupt on <enter> Port 1
n = 2	Interrupt on any character Port 1
n = -1	Clear any characters in buffer
m = 0	Disable entry mode
m = 1	Enable entry mode

The #COMINT label is used for the communication interrupt. For example, the controller can be configured to interrupt on any character received on Port 1. The #COMINT subroutine is entered when a character is received and the subroutine can decode the characters. At the end of the routine the EN command is used. EN,1 will re-enable the interrupt and return to the line of the program where the interrupt was called, EN will just return to the line of the program where it was called without re-

enabling the interrupt. As with any automatic subroutine, a program must be running in thread 0 at all times for it to be enabled.

Example

A controller is used to jog the A and B axis. This program allows the user to input values from the main serial port terminal. The speed of either axis may be changed during motion by specifying the axis letter followed by the new speed value. An ST stops motion on both axes.

```
#AUTO                                   Label for Auto Execute
speedA=10000                            Initial A speed
speedB=10000                            Initial B speed
CI 2,1                                  Set Port 1 for Character Interrupt
JG speedA, speedB                       Specify jog mode speed for A and B axis
BGXY                                    Begin motion
#PRINT                                  Routine to print message to terminal
MG"TO CHANGE SPEEDS"                    Print message
MG"TYPE A OR B"
MG"TYPE S TO STOP"
#JOGLOOP                                Loop to change Jog speeds
JG speedA, speedB                       Set new jog speed
JP #JOGLOOP
EN                                      End of main program
#COMINT                                 Interrupt routine
JP #A,P1CH="A"                          Check for A
JP #B,P1CH="B"                          Check for B
JP #C,P1CH="S"                          Check for S
ZS1;CI2;JP#JOGLOOP                      Jump if not X,Y,S
#A;JS#NUM
speedA=val                              New A speed
ZS1;CI2;JP#PRINT                        Jump to Print
#B;JS#NUM
speedB=val                              New B speed
ZS1;CI2;JP#PRINT                        Jump to Print
#C;ST;AMX;CI-1                          Stop motion on S
MG{^8}, "THE END"
ZS;EN,1                                 End-Re-enable interrupt
#NUM                                    Routine for entering new jog speed
MG "ENTER",P1CH{S},"AXIS SPEED"         Prompt for value
{N}
#NUMLOOP; CI-1                          Check for enter
#NMLP                                   Routine to check input from terminal
JP #NMLP,P1CD<2                         Jump to error if string
JP #ERROR,P1CD=2                        Read value
val=P1NM
EN                                      End subroutine
#ERROR;CI-1                             Error Routine
MG "INVALID-TRY AGAIN"                  Error message
```

```
JP #NMLP
EN                                  End
```

String Variables

String variables with up to six characters may be input using the specifier, {Sn} where n represents the number of string characters to be input. If n is not specified, six characters will be accepted. For example, IN "Enter A,B or C", V{S} specifies a string variable to be input.

The controller, stores all variables as 6 bytes of information. When a variable is specified as a number, the value of the variable is represented as 4 bytes of integer and 2 bytes of fraction. When a variable is specified as a string, the variable can hold up to 6 characters (each ASCII character is 1 byte). When using the IN command for string input, the first input character will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations, see section Bit-wise Operators.

Output of Data (Numeric and String)

Numerical and string data can be output from the controller using several methods. The message command, MG, can output string and numerical data. Also, the controller can be commanded to return the values of variables and arrays, as well as other information using the interrogation commands (the interrogation commands are described in chapter 5).

Sending Messages

Messages may be sent to the bus using the message command, MG. This command sends specified text and numerical or string data from variables or arrays to the screen.

Text strings are specified in quotes and variable or array data is designated by the name of the variable or array. For example:

```
MG "The Final Value is", result
```

In addition to variables, functions and commands, responses can be used in the message command. For example:

```
MG "Input  1 is", @IN[1]
MG "The Position of A is", _TPA
```

Formatting Messages

String variables can be formatted using the specifier, {Sn} where n is the number of characters, 1 thru 6. For example:

```
MG STR {S3}
```

This statement returns 3 characters of the string variable named STR.

Numeric data may be formatted using the {Fn.m} expression following the completed MG statement. {$n.m} formats data in HEX instead of decimal. The actual numerical value will be formatted with n characters to the left of the decimal and m characters to the right of the decimal. Leading zeros will be used to display specified format.

For example::

```
MG "The Final Value is", result {F5.2}
```

If the value of the variable result is equal to 4.1, this statement returns the following:

The Final Value is 00004.10

If the value of the variable result is equal to 999999.999, the above message statement returns the following:

The Final Value is 99999.99

The message command normally sends a carriage return and line feed following the statement. The carriage return and the line feed may be suppressed by sending {N} at the end of the statement. This is useful when a text string needs to surround a numeric value.

Example:

```
#A
JG 50000;BGA;ASA
MG "The Speed is", _TVA {F5.1} {N}
MG " counts/sec"
EN
```

When #A is executed, the above example will appear on the screen as:

The speed is 50000 counts/sec

Using the MG Command to Configure Terminals

The MG command can be used to configure a terminal. Any ASCII character can be sent by using the format {^n} where n is any integer between 1 and 255.

Example:

MG {^07} {^255}

sends the ASCII characters represented by 7 and 255 to the Ethernet/serial port.

Summary of Message Functions

function	description
" "	Surrounds text string
{Fn.m}	Formats numeric values in decimal n digits to the left of the decimal point and m digits to the right
{P1} or {En}	Send message to Serial Port or Ethernet Port
{$n.m}	Formats numeric values in hexadecimal
{^n}	Sends ASCII character specified by integer n
{N}	Suppresses carriage return/line feed
{Sn}	Sends the first n characters of a string variable, where n is 1 thru 6.

Displaying Variables and Arrays

Variables and arrays may be sent to the screen using the format, variable= **or** array[x]=. For example, v1= , returns the value of v1.

Example - Printing a Variable and an Array element

Instruction	Interpretation
#DISPLAY	Label
DM posA[7]	Define Array POSA with 7 entries

PR 1000	Position Command
BGX	Begin
AMX	After Motion
v1=_TPA	Assign Variable v1
posA[1]=_TPA	Assign the first entry
v1=	Print v1

Interrogation Commands

The CDS-3310 has a set of commands that directly interrogate the controller. When these command are entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), and Leading Zeros (LZ) command. For a complete description of interrogation commands, see Chapter 5.

Using PF to Format Response from Interrogation Commands

The command, PF, can change format of the values returned by theses interrogation commands:

BL ?	LE ?
DE ?	PA ?
DP ?	PR ?
FL ?	TN ?
IP ?	VE ?
TP	TE

The numeric values may be formatted in decimal or hexadecimal* with a specified number of digits to the right and left of the decimal point using the PF command.

Position Format is specified by:

PF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4) A negative sign for m specifies hexadecimal format.

*Hex values are returned preceded by a $ and in 2's complement. Hex values should be input as signed 2's complement, where negative numbers have a negative sign. The default format is PF 10.0.

If the number of decimal places specified by PF is less than the actual value, a nine appears in all the decimal places.

Example

Instruction	Interpretation
:DP21	Define position
:TPA	Tell position
0000000021	Default format
:PF4	Change format to 4 places
:TPA	Tell position
0021	New format
:PF-4	Change to hexadecimal format
:TPA	Tell Position
$0015	Hexadecimal value
:PF2	Format 2 places

:TPA	Tell Position
99	Returns 99 if position greater than 99

Removing Leading Zeros from Response to Interrogation Commands

The leading zeros on data returned as a response to interrogation commands can be removed by the use of the command, LZ.

LZ0	Disables the LZ function
TP	Tell Position Interrogation Command
-0000000009, 0000000005	Response (With Leading Zeros)
LZ1	Enables the LZ function
TP	Tell Position Interrogation Command
-9, 5	Response (Without Leading Zeros)

Local Formatting of Response of Interrogation Commands

The response of interrogation commands may be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} on the same line as the interrogation command. The symbol F specifies that the response should be returned in decimal format and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

TP {F2.2}	Tell Position in decimal format 2.2
-05.00, 05.00, 00.00, 07.00	Response from Interrogation Command
TP {$4.2}	Tell Position in hexadecimal format 4.2
FFFB.00,$0005.00,$0000.00,$0007.00	Response from Interrogation Command

Formatting Variables and Array Elements

The Variable Format (VF) command is used to format variables and array elements. The VF command is specified by:

VF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4).

A negative sign for m specifies hexadecimal format. The default format for VF is VF 10.4

Hex values are returned preceded by a $ and in 2's complement.

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1
:0000000010.0000	Response - Default format
VF2.2	Change format
v1=	Return v1
:10.00	Response - New format
vF-2.2	Specify hex format
v1=	Return v1
$0A.00	Response - Hex value
VF1	Change format
v1=	Return v1
:9	Response - Overflow

Local Formatting of Variables

PF and VF commands are global format commands that effect the format of all relevent returned values and variables. Variables may also be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} following the variable name and the '=' symbol. F specifies decimal and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1
:0000000010.0000	Default Format
v1={F4.2}	Specify local format
:0010.00	New format
v1={$4.2}	Specify hex format
:$000A.00	Hex value
v1="ALPHA"	Assign string "ALPHA" to v1
v1={S4}	Specify string format first 4 characters
:ALPH	

The local format is also used with the MG command.

Converting to User Units

Variables and arithmetic operations make it easy to input data in desired user units such as inches or RPM.

The CDS-3310 position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec^2. The controller interprets time in milliseconds.

All input parameters must be converted into these units. For example, an operator can be prompted to input a number in revolutions. A program could be used such that the input number is converted into counts by multiplying it by the number of counts/revolution.

Instruction	Interpretation
#RUN	Label
IN "ENTER # OF REVOLUTIONS",n1	Prompt for revs
PR n1*2000	Convert to counts
IN "ENTER SPEED IN RPM",s1	Prompt for RPMs
SP s1*2000/60	Convert to counts/sec
IN "ENTER ACCEL IN RAD/SEC2",a1	Prompt for accelleration
AC a1*2000/(2*3.14)	Convert to counts/sec^2
BG	Begin motion
EN	End program

Hardware I/O

Digital Outputs

The CDS-3310 has an 10-bit uncommitted output port and an additional 40 I/O (with the addition of the daughter board DB-28040) which may be configured as inputs or outputs with the CO command for controlling external events. Each bit on the output port may be set and cleared with the software instructions SB (Set Bit) and CB(Clear Bit), or OB (define output bit).

Example- Set Bit and Clear Bit

Instruction	Interpretation
SB6	Sets bit 6 of output port
CB4	Clears bit 4 of output port

Example- Output Bit

The Output Bit (OB) instruction is useful for setting or clearing outputs depending on the value of a variable, array, input or expression. Any non-zero value results in a set bit.

Instruction	Interpretation
OB1, POS	Set Output 1 if the variable POS is non-zero. Clear Output 1 if POS equals 0.
OB 2, @IN [1]	Set Output 2 if Input 1 is high. If Input 1 is low, clear Output 2.
OB 3, @IN [1]&@IN [2]	Set Output 3 only if Input 1 and Input 2 are high.
OB 4, COUNT [1]	Set Output 4 if element 1 in the array COUNT is non-zero.

The output port can be set by specifying an 8-bit word using the instruction OP (Output Port). This instruction allows a single command to define the state of the entire 8-bit output port, where 2^0 is output 1, 2^1 is output 2 and so on. A 1 designates that the output is on.

Example- Output Port

Instruction	Interpretation
OP6	Sets outputs 2 and 3 of output port to high. All other bits are 0. ($2^1 + 2^2 = 6$)
OP0	Clears all bits of output port to zero
OP 255	Sets all bits of output port to one. ($2^2 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7$)

The output port is useful for setting relays or controlling external switches and events during a motion sequence.

Example - Turn on output after move

PR 2000	Position Command
BG	Begin
AM	After move
SB1	Set Output 1
WT 1000	Wait 1000 msec
CB1	Clear Output 1
EN	End

Digital Inputs

The general digital inputs for are accessed by using the @IN[n] function or the TI command. The @IN[n] function returns the logic level of the specified input, n, where n is a number 1 through 96.

Example - Using Inputs to control program flow

Instruction	Interpretation
JP #A,@IN[1]=0	Jump to A if input 1 is low
JP #B,@IN[2]=1	Jump to B if input 2 is high
AI 7	Wait until input 7 is high
AI -6	Wait until input 6 is low

Example - Start Motion on Switch

Motor A must turn at 4000 counts/sec when the user flips a panel switch to 'on'. When panel switch is turned to 'off' position, motor A must stop turning.

Solution: Connect panel switch to input 1 of CDS-3310. High on input 1 means switch is in 'on' position.

Instruction	Interpretation
#S;JG 4000	Set speed
AI 1;BGA	Begin after input 1 goes high
AI -1;STA	Stop after input 1 goes low
AMA;JP #S	After motion, repeat
EN;	

Analog Inputs

The controller has two analog inputs and provides eight analog inputs with the addition of the DB-28040. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 through 8. The resolution of the Analog-to-Digital conversion is 12 bits (16-bit optional – Consult Galil). Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

The following examples show programs which cause the motor to follow an analog signal. The first example is a point-to-point move. The second example shows a continuous move.

Example - Position Follower (Point-to-Point)

Objective - The motor must follow an analog signal. When the analog signal varies by 10V, motor must move 10000 counts.

Method: Read the analog input and command A to move to that point.

Instruction	Interpretation
#POINTS	Label
SP 7000	Speed
AC 80000;DC 80000	Acceleration
#LOOP	
vp=@AN[1]*1000	Read and analog input, compute position
PA vp	Command position
BGA	Start motion

AMA	After completion
JP #LOOP	Repeat
EN	End

Example - Position Follower (Continuous Move)

Method: Read the analog input, compute the commanded position and the position error. Command the motor to run at a speed in proportions to the position error.

Instruction	Interpretation
#CONT	Label
AC 80000;DC 80000	Acceleration rate
JG 0	Start Jog mode
BGX	Start motion
#LOOP	
vp=@AN[1]*1000	Compute desired position, 1000 chosen for system
ve=vp-_TPA	Find position error
vel=ve*20	Compute velocity, 20 chosen for system
JG vel	Change velocity
JP #LOOP	Change velocity
EN	End

NOTE: Position tacking mode also provides for a continuous move.

Analog Output

The controller has one analog output which may be set in Volts using the AO 1,n command where n is the voltage from -9.999 to 9.999. The resolution of the Digital-to-Analog conversion is 16 bits.

Example

Objective – Produce a sinusoidal analog signal with a specified amplitude and period.

Instruction	Interpretation
Amplitud = 5	;'-5 Volts to +5 Volts
Period = 1000	;'1 second period
#loop	;'Label
AO 1, Amplitud * @SIN[360/Period * TIME]	;'set analog output
JP#loop	

The Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. The controller has one auxiliary encoder, which consists of two inputs, channel A and channel B. The auxiliary encoder inputs are mapped to the inputs 81-82.

Each input from the auxiliary encoder is a differential line receiver and can accept voltage levels between +/-12Volts. The inputs have been configured to accept TTL level signals. To connect TTL signals, simply connect the signal to the + input and leave the - input disconnected. For other signal levels, the - input should be connected to a voltage that is ½ of the full voltage range (for example, connect the - input to 6 volts if the signal is a 0 - 12 volt logic).

Example:

A CDS-3310 has one auxiliary encoder. This encoder has two inputs (channel A and channel B). Channel A input is mapped to input 81 and Channel B input is mapped to input 82. To use this input for 2 TTL signals, the first signal will be connected to AA+ and the second to AB+. AA- and AB- will be left unconnected. To access this input, use the function @IN[81] and @IN[82].

Input Interrupt Function

The CDS-3310 provides an input interrupt function which causes the program to automatically execute the instructions following the #ININT label. This function is enabled using the II m,n,o command. The m specifies the beginning input and n specifies the final input in the range. The parameter o is an interrupt mask. If m and n are unused, o contains a number with the mask. A 1 designates that input to be enabled for an interrupt, where 2^0 is bit 1, 2^1 is bit 2 and so on. For example, II,,5 enables inputs 1 and 3 ($2^0 + 2^2 = 5$).

A low input on any of the specified inputs will cause automatic execution of the #ININT subroutine. The Return from Interrupt (RI) command is used to return from this subroutine to the place in the program where the interrupt had occurred. If it is desired to return to somewhere else in the program after the execution of the #ININT subroutine, the Zero Stack (ZS) command is used followed by unconditional jump statements.

Important: Use the RI command (not EN) to return from the #ININT subroutine.

Example - Input Interrupt

Instruction	Interpretation
#A	Label #A
II 1	Enable input 1 for interrupt function
JG 30000,-20000	Set speeds on A and B axes
BG AB	Begin motion on A and B axes
#B	Label #B
TP AB	Report A and B axes positions
WT 1000	Wait 1000 milliseconds
JP #B	Jump to #B
EN	End of program
#ININT	Interrupt subroutine
MG "Interrupt has occurred"	Displays the message
ST AB	Stops motion on A and B axes
#LOOP;JP #LOOP,@IN[1]=0	Loop until Interrupt cleared
JG 15000,10000	Specify new speeds
WT 300	Wait 300 milliseconds
BG AB	Begin motion on A and B axes
RI	Return from Interrupt subroutine

Extended I/O of the CDS-3310 Controller

The CDS-3310 controller, with optional DB-28040, offers 40 extended I/O points which can be configured as inputs or outputs in 8 bit increments through software. The I/O points are accessed through the daughter board, which utilizes one 50-pin IDC ribbon connectors.

Configuring the I/O of the CDS-3310

The 40 extended I/O points of the CDS-3310 series controller can be configured in banks of 8. The extended I/O is denoted as banks 2-6 or bits 17-56.

The command, CO, is used to configure the extended I/O as inputs or outputs. The CO command has one field:

CO n

where n is a decimal value which represents a binary number. Each bit of the binary number represents one bank of extended I/O. When set to 1, the corresponding bank is configured as an output.

The least significant bit represents bank 2 and the most significant bit represents bank 9. The decimal value can be calculated by the following formula. $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 + 16* n_6$ where n_x represents the bank. If the n_x value is a one, then the bank of 8 I/O points is to be configured as an output. If the n_x value is a zero, then the bank of 8 I/O points will be configured as an input. For example, if bank 4 and 5 is to be configured as an output, CO 12 is issued.

8-Bit I/O Bank	Bank	Binary Representation	Decimal Value for Bank
17-24	2	2^0	1
25-32	3	2^1	2
33-40	4	2^2	4
41-48	5	2^3	8
49-56	6	2^4	16

The simplest method for determining n:

Step 1. Determine which 8-bit I/O banks to be configured as outputs.

Step 2. From the table, determine the decimal value for each I/O bank to be set as an output.

Step 3. Add up all of the values determined in step 2. This is the value to be used for n.

For example, if banks 2 and 3 are to be outputs, then n is 3 and CO3 should be issued. **NOTE:** This calculation is identical to the formula: $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 + 16* n_6$ where n_x represents the bank.

Saving the State of the Outputs in Non-Volatile Memory

The configuration of the extended I/O and the state of the outputs can be stored in the EEPROM with the BN command. If no value has been set, the default of CO 0 is used (all banks are inputs).

Accessing Extended I/O

When configured as an output, each I/O point may be defined with the SBn and CBn commands (where n=1 through 8 and 17 through 56). Outputs may also be defined with the conditional command, OBn (where n=1 through 8 and 17 through 56).

The command, OP, may also be used to set output bits, specified as banks of data. The OP command accepts 5 parameters. The first parameter sets the values of the main output port of the controller (Outputs 1-8, bank 0). The additional parameters set the value of the extended I/O as outlined:

OP m,a,b,c,d

where m is the decimal representation of the bits 1-8 (values from 0 to 255) and a,b,c,d represent the extended I/O in consecutive groups of 16 bits. (values from 0 to 65535). Arguments which are given for I/O points which are configured as inputs will be ignored. The following table describes the arguments used to set the state of outputs.

Argument	Banks	Bits	Description
m	0	1-8	General Outputs
a	2,3	17-32	Extended I/O
b	4,5	33-48	Extended I/O
c	6,7	49-56	Extended I/O

For example, the following command may be issued:

OP 7

This command will set bits 1,2,3 (bank 0) to 1. Bits 4 through 8 will be set to 0. All other bits are unaffected.

When accessing I/O banks configured as inputs, use the TIn command. The argument 'n' refers to the bank to be read (n=0,2,3,4,5,6). The value returned will be a decimal representation of the corresponding bits.

Individual bits can be queried using the @IN[n] function (where n=1 through 8 or 17 through 56). If the following command is issued;

MG @IN[17]

the controller will return the state of the least significant bit of bank 2 (assuming bank 2 is configured as an input).

Example Applications

Wire Cutter

An operator activates a start switch. This causes a motor to advance the wire a distance of 10". When the motion stops, the controller generates an output signal which activates the cutter. Allowing 100 ms for the cutting completes the cycle.

Suppose that the motor drives the wire by a roller with a 2" diameter. Also assume that the encoder resolution is 1000 lines per revolution. Since the circumference of the roller equals 2π inches, and it corresponds to 4000 quadrature, one inch of travel equals:

$4000/2\pi = 637$ count/inch

This implies that a distance of 10 inches equals 6370 counts, and a slew speed of 5 inches per second, for example, equals 3185 count/sec.

The input signal may be applied to I1, for example, and the output signal is chosen as output 1. The motor velocity profile and the related input and output signals are shown in Fig. 7.1.

The program starts at a state that we define as #A. Here the controller waits for the input pulse on I1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to #A for a new cycle.

Instruction	Interpretation
#A	Label
AI1	Wait for input 1
PR 6370	Distance
SP 3185	Speed
BGA	Start Motion
AMA	After motion is complete
SB1	Set output bit 1
WT 20	Wait 20 ms
CB1	Clear output bit 1
WT 80	Wait 80 ms
JP #A	Repeat the process

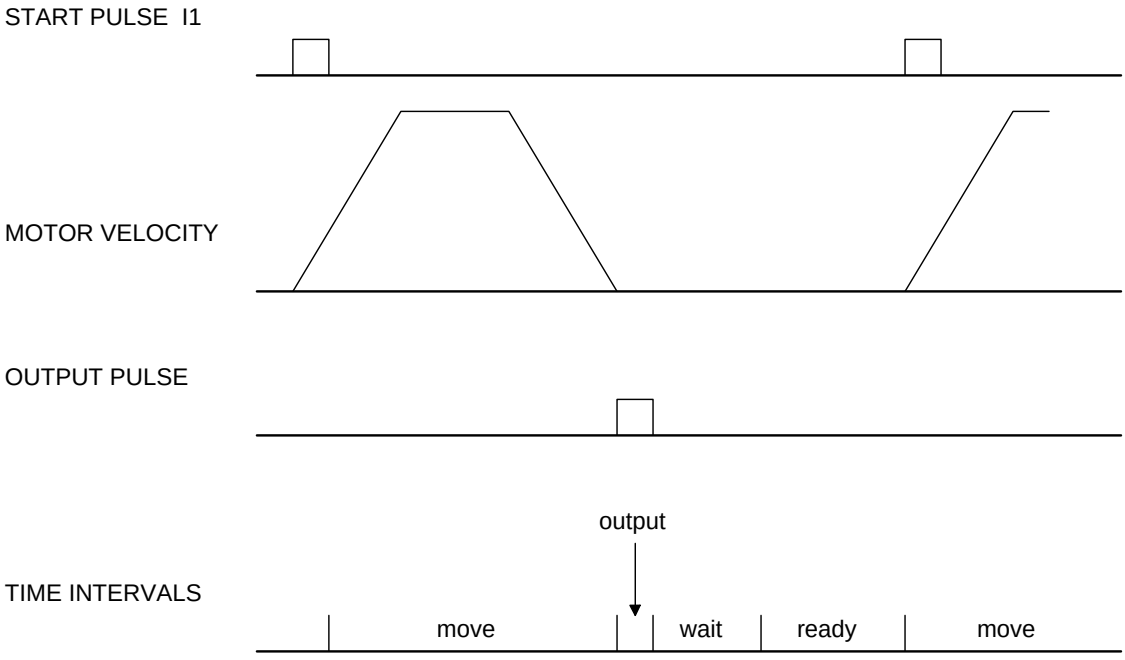


Figure 7.1 - Motor Velocity and the Associated Input/Output signals

Backlash Compensation by Sampled Dual-Loop

The continuous dual loop, enabled by the DV1 function is an effective way to compensate for backlash. In some cases, however, when the backlash magnitude is large, it may be difficult to stabilize the system. In those cases, it may be easier to use the sampled dual loop method described below.

This design example addresses the basic problems of backlash in motion control systems. The objective is to control the position of a linear slide precisely. The slide is to be controlled by a rotary motor, which is coupled to the slide by a leadscrew. Such a leadscrew has a backlash of 4 micron, and the required position accuracy is for 0.5 micron.

The basic dilemma is where to mount the sensor. If you use a rotary sensor, you get a 4 micron backlash error. On the other hand, if you use a linear encoder, the backlash in the feedback loop will cause oscillations due to instability.

An alternative approach is the dual-loop, where we use two sensors, rotary and linear. The rotary sensor assures stability (because the position loop is closed before the backlash) whereas the linear sensor provides accurate load position information. The operation principle is to drive the motor to a given rotary position near the final point. Once there, the load position is read to find the position error and the controller commands the motor to move to a new rotary position which eliminates the position error.

Since the required accuracy is 0.5 micron, the resolution of the linear sensor should preferably be twice finer. A linear sensor with a resolution of 0.25 micron allows a position error of +/-2 counts.

The dual-loop approach requires the resolution of the rotary sensor to be equal or better than that of the linear system. Assuming that the pitch of the lead screw is 2.5mm (approximately 10 turns per inch), a rotary encoder of 2500 lines per turn or 10,000 count per revolution results in a rotary resolution of 0.25 micron. This results in equal resolution on both linear and rotary sensors.

To illustrate the control method, assume that the rotary encoder is used as a feedback for the X-axis, and that the linear sensor is read and stored in the variable LINPOS. Further assume that at the start, both the position of X and the value of LINPOS are equal to zero. Now assume that the objective is to move the linear load to the position of 1000.

The first step is to command the X motor to move to the rotary position of 1000. Once it arrives we check the position of the load. If, for example, the load position is 980 counts, it implies that a correction of 20 counts must be made. However, when the X-axis is commanded to be at the position of 1000, suppose that the actual position is only 995, implying that X has a position error of 5 counts, which will be eliminated once the motor settles. This implies that the correction needs to be only 15 counts, since 5 counts out of the 20 would be corrected by the X-axis. Accordingly, the motion correction should be:

Correction = Load Position Error - Rotary Position Error

The correction can be performed a few times until the error drops below +/-2 counts. Often, this is performed in one correction cycle.

Instruction	Interpretation
#A	Label
DP0	Define starting positions as zero
linpos=0	
PR 1000	Required distance
BGA	Start motion
#B	
AMA	Wait for completion
WT 50	Wait 50 msec
linpos = _DEA	Read linear position
er=1000- linpos -_TEA	Find the correction
JP #C,@ABS[er]<2	Exit if error is small
PR er	Command correction
BGA	
JP #B	Repeat the process

Chapter 8 Hardware & Software Protection

Introduction

The CDS-3310 provides several hardware and software features to check for error conditions and to inhibit the motor on error. These features help protect the various system components from damage.

***Note**: *Do not "hot swap" the motor power connections. If the amp is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector.

WARNING: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machine. Since the CDS-3310 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the CDS-3310. Galil shall not be liable or responsible for any incidental or consequential damages.

Hardware Protection

The CDS-3310 includes hardware input and output protection lines for various error and mechanical limit conditions. These include:

Output Protection Lines

Error Output - The error output is a TTL signal which indicates on error condition in the controller. This signal is available on the interconnect module as ERROR. When the error signal is low, this indicates on of the following error conditions.

1. At least one axis has a position error greater than the error limit. The error limit is set by using the command ER.
2. The reset line on the controller is held low or is being affected by noise.
3. There is a failure on the controller and the processor is resetting itself.
4. There is a failure with the output IC which drives the error signal.

Input Protection Lines

General Abort- A low input stops commanded motion instantly without a controlled deceleration. For any axis in which the Off On Error function is enabled, the amplifiers will be disabled. This could cause the motor to 'coast' to a stop. If the Off On Error function is not enabled, the motor will instantaneously stop and servo at the current position. The Off On Error function is further discussed in this chapter.

Forward Limit Switch - Low input inhibits motion in forward direction. If the motor is moving in the forward direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the forward direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches.

Reverse Limit Switch - Low input inhibits motion in reverse direction. If the motor is moving in the reverse direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the reverse direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches.

Amplifier Status LED's

Over-Voltage Protection – The CDS-3310 is protected against over voltage. If the supply voltage exceeds 78 V, the over voltage protection will take effect. The yellow over voltage LED will be lit on the amplifier until the voltage drops below 74 V. This error is monitored with bit 1 of the TA0 command.

It is possible to get into this condition if the power supply voltage is too high, or if the voltage level is raised due to regeneration. If you have very high inertial loads (which may cause regeneration), you may consider using a shunt regulator such as the SR-199000 supplied by Galil.

Over-Current Protection – The controller also has protection against over current. If the total current from the supply exceeds 15 A, the amplifier will be disabled. It can be enabled again from the controller by issuing the MO and then SH command. The amplifier will be shut down regardless of the setting of OE, or the presence of the #AMPERR routine. Bit 0 of TA0 will be set. If you see that the red over current LED is lit on the amplifier, there is a problem with either your system or the amplifier. The most likely reason is because of a short between the motor phases or between the motor phases and ground. This indicates either a wiring problem, or a faulty motor.

Software Protection

The CDS-3310 provides a programmable error limit. The error limit can be set for any number between 1 and 32767 using the ER n command. The default value for ER is 16384.

ER 200,300,400,500	Set A-axis error limit for 200, B-axis error limit to 300, C-axis error limit to 400 counts, D-axis error limit to 500 counts
ER,1,,10	Set B-axis error limit to 1 count, set D-axis error limit to 10 counts.

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the CDS-3310 will generate several signals to warn the host system of the error condition. These signals include:

SIGNAL OR FUNCTION	STATE IF ERROR OCCURS
# POSERR	Jumps to automatic excess position error subroutine
Error Light	Turns on
OE Function	Shuts motor off if OE1
AEN Output Line	Goes low

The Jump on Condition statement is useful for branching on a given error within a program. The position error of A,B,C and D can be monitored during execution using the TE command.

Programmable Position Limits

The CDS-3310 provides programmable forward and reverse position limits. These are set by the BL and FL software commands. Once a position limit is specified, the CDS-3310 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

Example

DP0,0,0	Define Position
BL -2000,-4000,-8000	Set Reverse position limit
FL 2000,4000,8000	Set Forward position limit
JG 2000,2000,2000	Jog
BG ABC	Begin (motion stops at forward limits)

Off-On-Error

The CDS-3310 controller has a built in function which can turn off the motors under certain error conditions. This function is know as 'Off-On-Error". To activate the OE function for each axis, specify 1 for A,B,C and D axis. To disable this function, specify 0 for the axes. When this function is enabled, the specified motor will be disabled under the following 3 conditions:

1. The position error for the specified axis exceeds the limit set with the command, ER
2. The abort command is given
3. The abort input is activated with a low signal.

NOTE: If the motors are disabled while they are moving, they may 'coast' to a stop because they are no longer under servo control.

To re-enable the system, use the Reset (RS) or Servo Here (SH) command.

Example

OE 1,1,1,1	Enable off on error for A,B,C and D
OE 0,1,0,1	Enable off on error for B and D axes, Disable off on error for A and C

Automatic Error Routine

The #POSERR label causes the statements following to be automatically executed if error on any axis exceeds the error limit specified by ER. The error routine must be closed with the RE command. The RE command returns from the error subroutine to the main program.

NOTE: The Error Subroutine will be entered again unless the error condition is gone.

Example

Instruction	Interpretation
#A;JP #A;EN	"Dummy" program
#POSERR	Start error routine on error
MG "error"	Send message
SB 1	Fire relay
STA	Stop motor
AMA	After motor stops
SHA	Servo motor here to clear error
RE	Return to main program

NOTE: An applications program must be executing for the #POSERR routine to function.

Limit Switch Routine

The CDS-3310 provides forward and reverse limit switches which inhibit motion in the respective direction. There is also a special label for automatic execution of a limit switch subroutine. The #LIMSWI label specifies the start of the limit switch subroutine. This label causes the statements following to be automatically executed if any limit switch is activated and that axis motor is moving in that direction. The RE command ends the subroutine.

The state of the forward and reverse limit switches may also be tested during the jump-on-condition statement. The _LR condition specifies the reverse limit and _LF specifies the forward limit. A,B,C, or D following LR or LF specifies the axis. The CN command can be used to configure the polarity of the limit switches.

Example

Instruction	Interpretation
#A;JP #A;EN	Dummy Program
#LIMSWI	Limit Switch Utility
v1=_LFA	Check if forward limit
v2=_LRA	Check if reverse limit
JP#LF,v1=0	Jump to #LF if forward
JP#LR,v2=0	Jump to #LR if reverse
JP#END	Jump to end
#LF	#LF
MG "FORWARD LIMIT"	Send message
STX;AMA	Stop motion
PR-1000;BGA;AMA	Move in reverse
JP#END	End
#LR	#LR
MG "REVERSE LIMIT"	Send message
STX;AMA	Stop motion
PR1000;BGA;AMA	Move forward
#END	End
RE	Return to main program

NOTE: An applications program must be executing for #LIMSWI to function.

Amplifier Error Routine

If there is an overcurrent, overvoltage, undervoltage, or Hall input error, the application program will jump to the #AMPERR automatic subroutine. Once the routine is running, the user can program conditional statements to alert the operator to the amplifier error, shut off the amplifiers, turn on the error output, etc.

Instruction	Interpretation
#A;JP #A;EN	Dummy Program
#AMPERR	Amplifier Error Utility
v1=_TA1;v2=_TA2	Set variable v1 to general amplifier; set v2 to Hall Error
IF (v2=1)	Check Hall error on X axis
MG"Hall Error on X"	Alert Operator
ENDIF	
IF (v2=2)	Check Hall error on Y axis
MG"Hall Error on Y"	Alert Operator
ENDIF	
IF (v2=4)	Check Hall error on Z axis
MG"Hall Error on Z"	Alert Operator
ENDIF	
If (v2=8)	Check Hall Error on W axis
MG"Hall Error on W"	Alert Operator
IF (v1=1)	Check for Overcurrent
MG"Overcurrent"	Alert Operator
ENDIF	
IF (v1=2)	Check for Overvoltage
MG"Overvoltage"	Alert Operator
ENDIF	
IF (v1=4)	Check for Overtemperature
MG"Too Hot!"	Alert Operator
ENDIF	
IF (v1=8)	Check for Undervoltage
MG" Voltage too low"	Alert Operator
ENDIF	
MO	Motors Off
EN	End Program

THIS PAGE LEFT PLANK INTENTIONALLY

Chapter 9 Troubleshooting

Overview

The following discussion may help you get your system to work.

Potential problems have been divided into groups as follows:

1. Installation
2. Communication
3. Stability and Compensation
4. Operation

The various symptoms along with the cause and the remedy are described in the following tables.

Installation

SYMPTOM	CAUSE	REMEDY
Motor runs away when connected to amplifier with no additional inputs.	Amplifier offset too large.	Adjust amplifier offset
Same as above, but offset adjustment does not stop the motor.	Damaged amplifier.	Repair amplifier.
Controller does not read changes in encoder position.	Wrong encoder connections.	Check encoder wiring.
Same as above	Bad encoder	Check the encoder signals. Replace encoder if necessary.
Same as above	Bad controller	Connect the encoder to different axis input. If it works, controller failure. Repair or replace.

Communication

SYMPTOM	CAUSE	REMEDY
Using terminal emulator, cannot communicate with controller.	Selected comm. port incorrect	Try another comport
Same as above	Selected baud rate incorrect	Check to be sure that baud rate same as jumper settings on controller, change as necessary.

Stability

SYMPTOM	CAUSE	REMEDY
Motor runs away when the loop is closed.	Wrong feedback polarity.	Invert the polarity of the loop by inverting the motor leads (brush type) or the encoder.
Motor oscillates.	Too high gain or too little damping.	Decrease KI and KP. Increase KD.

Operation

SYMPTOM	CAUSE	REMEDY
Controller rejects command. Responded with a ?	Anything.	Interrogate the cause with TC or TC1.
Motor does not complete move.	Noise on limit switches stops the motor.	To verify cause, check the stop code (SC). If caused by limit switch noise, reduce noise.
During a periodic operation, motor drifts slowly.	Encoder noise	Interrogate the position periodically. If controller states that the position is the same at different locations it implies encoder noise. Reduce noise. Use differential encoder inputs.
Same as above.	Programming error.	Avoid resetting position error at end of move with SH command.

Chapter 10 Theory of Operation

Overview

The following discussion covers the operation of motion control systems. A typical motion control system consists of the elements shown in Fig 10.1.

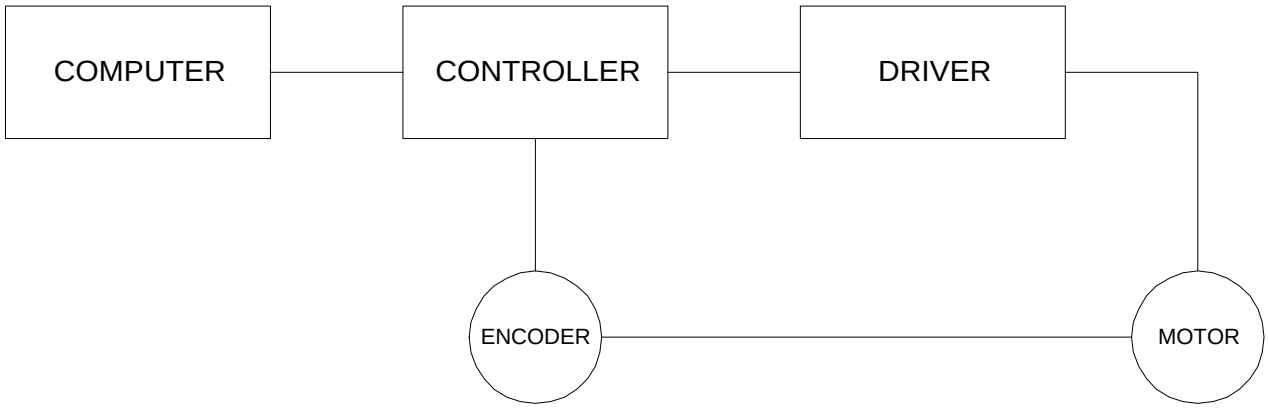


Figure 10.1 - Elements of Servo Systems

The operation of such a system can be divided into three levels, as illustrated in Fig. 10.2. The levels are:

1. Closing the Loop
2. Motion Profiling
3. Motion Programming

The first level, the closing of the loop, assures that the motor follows the commanded position. This is done by closing the position loop using a sensor. The operation at the basic level of closing the loop involves the subjects of modeling, analysis, and design. These subjects will be covered in the following discussions.

The motion profiling is the generation of the desired position function. This function, R(t), describes where the motor should be at every sampling period. Note that the profiling and the closing of the loop are independent functions. The profiling function determines where the motor should be and the closing of the loop forces the motor to follow the commanded position

The highest level of control is the motion program. This can be stored in the host computer or in the controller. This program describes the tasks in terms of the motors that need to be controlled, the distances and the speed.

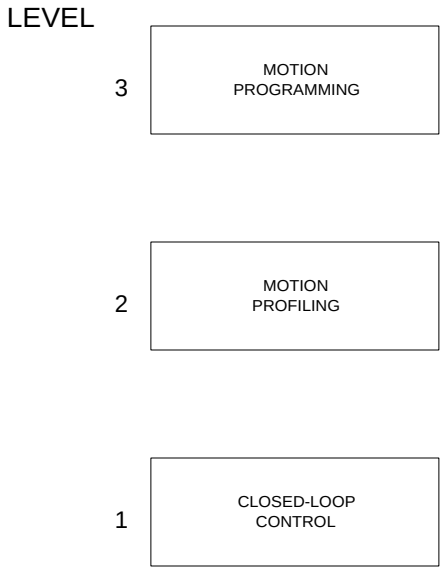


Figure 10.2 - Levels of Control Functions

The three levels of control may be viewed as different levels of management. The top manager, the motion program, may specify the following instruction, for example.

```
PR 6000,4000
SP 20000,20000
AC 200000,00000
BG A
AD 2000
BG B
EN
```

This program corresponds to the velocity profiles shown in Fig. 10.3. Note that the profiled positions show where the motors must be at any instant of time.

Finally, it remains up to the servo system to verify that the motor follows the profiled position by closing the servo loop.

The following section explains the operation of the servo system. First, it is explained qualitatively, and then the explanation is repeated using analytical tools for those who are more theoretically inclined.

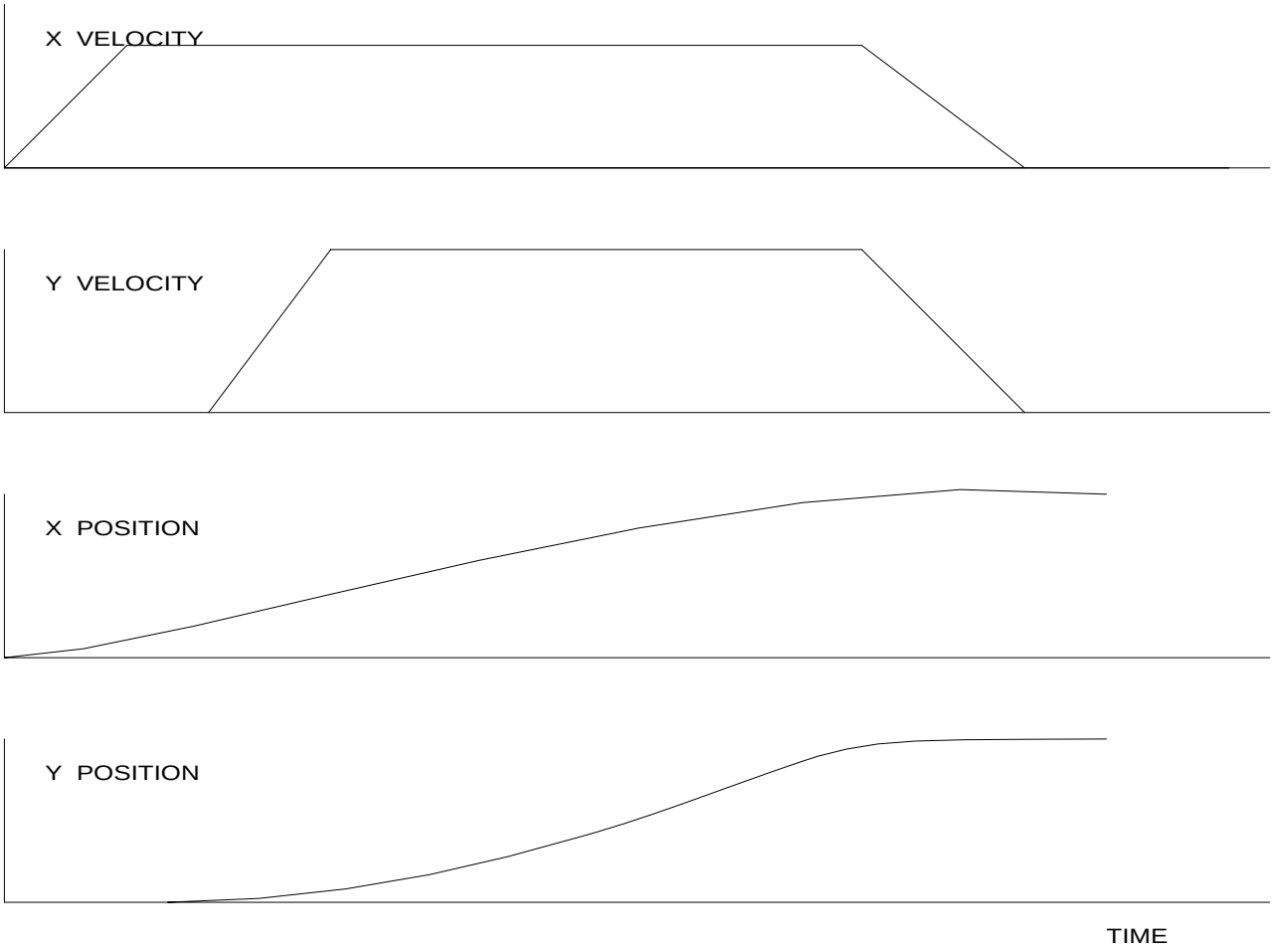


Figure 10.3 - Velocity and Position Profiles

Operation of Closed-Loop Systems

To understand the operation of a servo system, we may compare it to a familiar closed-loop operation, adjusting the water temperature in the shower. One control objective is to keep the temperature at a comfortable level, say 90 degrees F. To achieve that, our skin serves as a temperature sensor and reports to the brain (controller). The brain compares the actual temperature, which is called the feedback signal, with the desired level of 90 degrees F. The difference between the two levels is called the error signal. If the feedback temperature is too low, the error is positive, and it triggers an action which raises the water temperature until the temperature error is reduced sufficiently.

The closing of the servo loop is very similar. Suppose that we want the motor position to be at 90 degrees. The motor position is measured by a position sensor, often an encoder, and the position feedback is sent to the controller. Like the brain, the controller determines the position error, which is the difference between the commanded position of 90 degrees and the position feedback. The controller then outputs a signal that is proportional to the position error. This signal produces a proportional current in the motor, which causes a motion until the error is reduced. Once the error becomes small, the resulting current will be too small to overcome the friction, causing the motor to stop.

The analogy between adjusting the water temperature and closing the position loop carries further. We have all learned the hard way, that the hot water faucet should be turned at the "right" rate. If you turn it too slowly, the temperature response will be slow, causing discomfort. Such a slow reaction is called overdamped response.

The results may be worse if we turn the faucet too fast. The overreaction results in temperature oscillations. When the response of the system oscillates, we say that the system is unstable. Clearly, unstable responses are bad when we want a constant level.

What causes the oscillations? The basic cause for the instability is a combination of delayed reaction and high gain. In the case of the temperature control, the delay is due to the water flowing in the pipes. When the human reaction is too strong, the response becomes unstable.

Servo systems also become unstable if their gain is too high. The delay in servo systems is between the application of the current and its effect on the position. Note that the current must be applied long enough to cause a significant effect on the velocity, and the velocity change must last long enough to cause a position change. This delay, when coupled with high gain, causes instability.

This motion controller includes a special filter which is designed to help the stability and accuracy. Typically, such a filter produces, in addition to the proportional gain, damping and integrator. The combination of the three functions is referred to as a PID filter.

The filter parameters are represented by the three constants KP, KI and KD, which correspond to the proportional, integral and derivative term respectively.

The damping element of the filter acts as a predictor, thereby reducing the delay associated with the motor response.

The integrator function, represented by the parameter KI, improves the system accuracy. With the KI parameter, the motor does not stop until it reaches the desired position exactly, regardless of the level of friction or opposing torque.

The integrator also reduces the system stability. Therefore, it can be used only when the loop is stable and has a high gain.

The output of the filter is applied to a digital-to-analog converter (DAC). The resulting output signal in the range between +10 and -10 Volts is then applied to the amplifier and the motor.

The motor position, whether rotary or linear is measured by a sensor. The resulting signal, called position feedback, is returned to the controller for closing the loop.

The following section describes the operation in a detailed mathematical form, including modeling, analysis and design.

System Modeling

The elements of a servo system include the motor, driver, encoder and the controller. These elements are shown in Fig. 10.4. The mathematical model of the various components is given below.

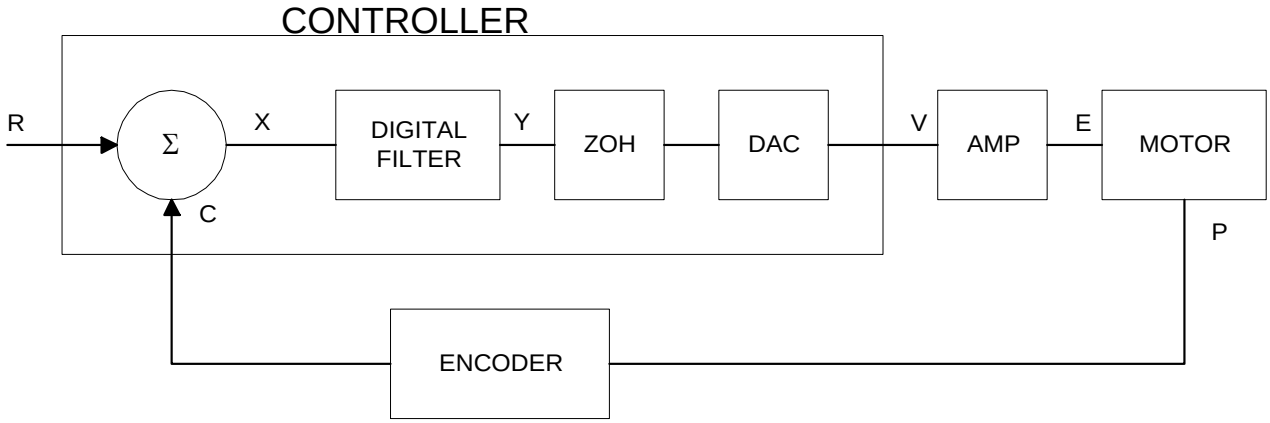


Figure 10.4 - Functional Elements of a Motion Control System

Motor-Amplifier

The current drive generates a current I, which is proportional to the input voltage, V, with a gain of Ka. The resulting transfer function in this case is

$$P/V = K_a K_t / Js^2$$

where Kt and J are as defined previously. For example, a current amplifier with K_a = 2 A/V with the motor described by the previous example will have the transfer function:

$$P/V = 1000/s^2 \qquad [rad/V]$$

If the motor is a DC brushless motor, it is driven by an amplifier that performs the commutation. The combined transfer function of motor amplifier combination is the same as that of a similar brush motor, as described by the previous equations.

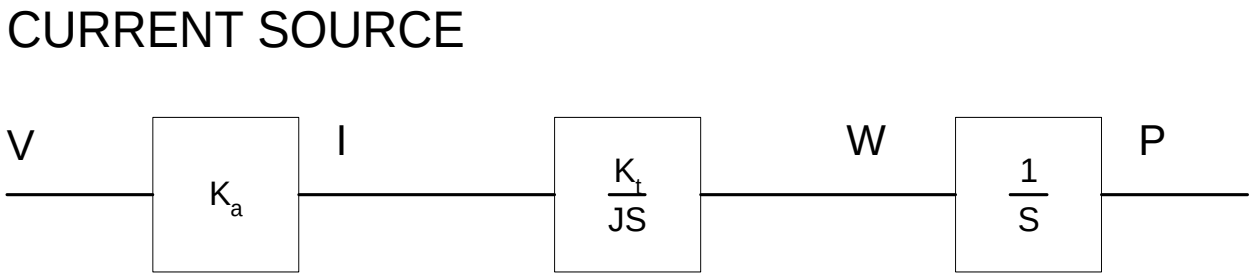


Figure 10.6 - Mathematical model of the motor and amplifier

Encoder

The encoder generates N pulses per revolution. It outputs two signals, Channel A and B, which are in quadrature. Due to the quadrature relationship between the encoder channels, the position resolution is increased to 4N quadrature counts/rev.

The model of the encoder can be represented by a gain of

$$K_f = 4N/2\pi \qquad [count/rad]$$

For example, a 1000 lines/rev encoder is modelled as

$$K_f = 638$$

DAC

The DAC or D-to-A converter converts a 16-bit number to an analog voltage. The input range of the numbers is 65536 and the output voltage range is +/-10V or 20V. Therefore, the effective gain of the DAC is

$$K = 20/65536 = 0.0003 \qquad [V/count]$$

Digital Filter

The digital filter has three element in series: PID, low-pass and a notch filter. The transfer function of the filter. The transfer function of the filter elements are:

PID $D(z) = \frac{K(Z-A)}{Z} + \frac{CZ}{Z-1}$

Low-pass $L(z) = \frac{1-B}{Z-B}$

Notch $N(z) = \frac{(Z-z)(Z-\bar{z})}{(Z-p)(Z-\bar{p})}$

The filter parameters, K, A, C and B are selected by the instructions KP, KD, KI and PL, respectively. The relationship between the filter coefficients and the instructions are:

K = (KP + KD) • 4

A = KD/(KP + KD)

C = KI/2

B = PL

The PID and low-pass elements are equivalent to the continuous transfer function G(s).

G(s) = (P + sD + I/s) ∗ a/(S+a)

P = 4KP

D = 4T • KD

I = KI/2T

a = 1/T ln (1/B)

where T is the sampling period.

For example, if the filter parameters of the CDS-3310 are

KP = 4

KD = 36

KI = 2

PL = 0.75

T = 0.001 s

the digital filter coefficients are

K = 160

A = 0.9

C = 1

a = 250 rad/s

and the equivalent continuous filter, G(s), is

G(s) = [16 + 0.144s + 1000/s} ∗ 250/ (s+250)

The notch filter has two complex zeros, Z and z, and two complex poles, P and p.

The effect of the notch filter is to cancel the resonance affect by placing the complex zeros on top of the resonance poles. The notch poles, P and p, are programmable and are selected to have sufficient

damping. It is best to select the notch parameters by the frequency terms. The poles and zeros have a frequency in Hz, selected by the command NF. The real part of the poles is set by NB and the real part of the zeros is set by NZ.

The most simple procedure for setting the notch filter, identify the resonance frequency and set NF to the same value. Set NB to about one half of NF and set NZ to a low value between zero and 5.

ZOH

The ZOH, or zero-order-hold, represents the effect of the sampling process, where the motor command is updated once per sampling period. The effect of the ZOH can be modeled by the transfer function

$$H(s) = 1/(1+sT/2)$$

If the sampling period is T = 0.001, for example, H(s) becomes:

$$H(s) = 2000/(s+2000)$$

However, in most applications, H(s) may be approximated as one.

This completes the modeling of the system elements. Next, we discuss the system analysis.

System Analysis

To analyze the system, we start with a block diagram model of the system elements. The analysis procedure is illustrated in terms of the following example.

Consider a position control system with the CDS-3310 controller and the following parameters:

$K_t = 0.1$	Nm/A	Torque constant
$J = 2.10^{-4}$	$kg.m^2$	System moment of inertia
R = 2	Ω	Motor resistance
$K_a = 4$	Amp/Volt	Current amplifier gain
KP = 12.5		Digital filter gain
KD = 245		Digital filter zero
KI = 0		No integrator
N = 500	Counts/rev	Encoder line density
T = 1	ms	Sample period

The transfer function of the system elements are:

Motor

$$M(s) = P/I = Kt/Js2 = 500/s^2 \ \ [rad/A]$$

Amp

$$K_a = 4 \ \ [Amp/V]$$

DAC

$$K_d = 0.0003 \ \ [V/count]$$

Encoder

$$K_f = 4N/2\pi = 318 \ [count/rad]$$

ZOH

$$2000/(s+2000)$$

Digital Filter

KP = 12.5, KD = 245, T = 0.001

Therefore,

D(z) = 1030 (z-0.95)/Z

Accordingly, the coefficients of the continuous filter are:

P = 50

D = 0.98

The filter equation may be written in the continuous equivalent form:

G(s) = 50 + 0.98s = .098 (s+51)

The system elements are shown in Fig. 10.7.

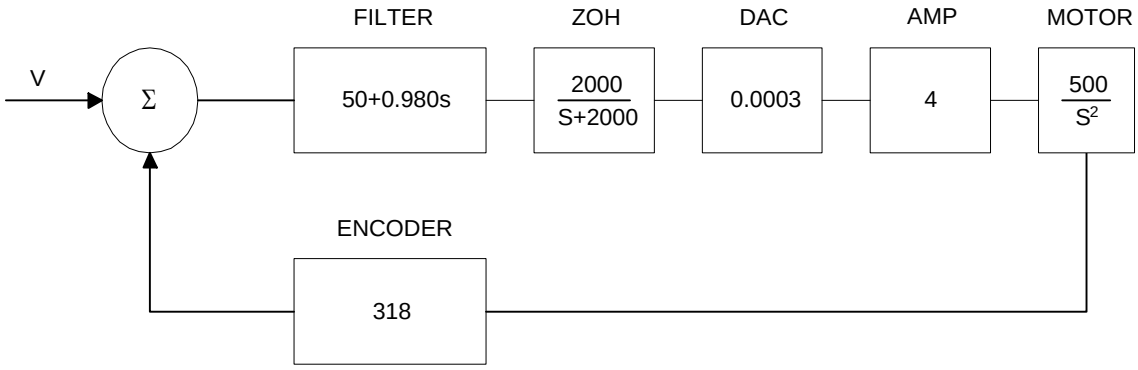


Figure 10.7 - Mathematical model of the control system

The open loop transfer function, A(s), is the product of all the elements in the loop.

$A = 390{,}000\ (s+51)/[s^2(s+2000)]$

To analyze the system stability, determine the crossover frequency, ω_c at which $A(j\ \omega_c)$ equals one. This can be done by the Bode plot of $A(j\ \omega_c)$, as shown in Fig. 10.8.

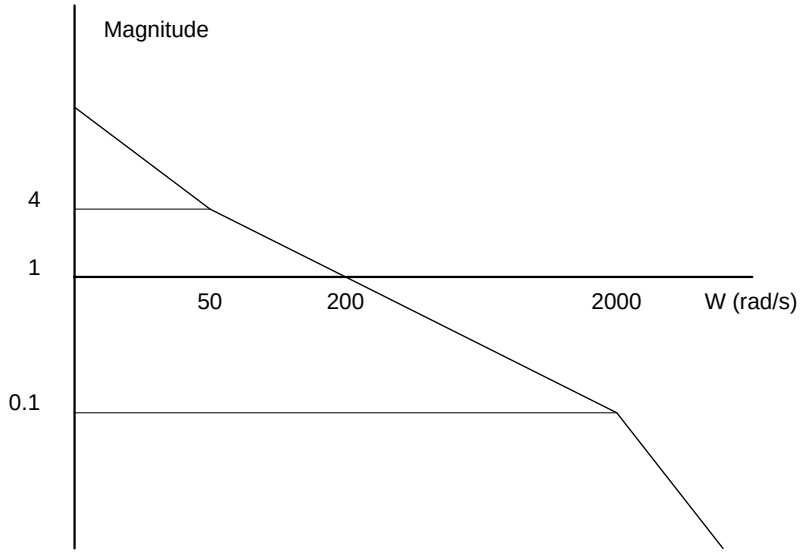


Figure 10.8 - Bode plot of the open loop transfer function

For the given example, the crossover frequency was computed numerically resulting in 200 rad/s.

Next, we determine the phase of A(s) at the crossover frequency.

$$A(j200) = 390{,}000\ (j200+51)/[(j200)^2\ .\ (j200 + 2000)]$$

$$\alpha = \text{Arg}[A(j200)] = \tan^{-1}(200/51) - 180° - \tan^{-1}(200/2000)$$

$$\alpha = 76° - 180° - 6° = -110°$$

Finally, the phase margin, PM, equals

$$PM = 180° + \alpha = 70°$$

As long as PM is positive, the system is stable. However, for a well damped system, PM should be between 30 degrees and 45 degrees. The phase margin of 70 degrees given above indicated overdamped response.

Next, we discuss the design of control systems.

System Design and Compensation

The closed-loop control system can be stabilized by a digital filter, which is preprogrammed in the CDS-3310 controller. The filter parameters can be selected by the user for the best compensation. The following discussion presents an analytical design method.

The Analytical Method

The analytical design method is aimed at closing the loop at a crossover frequency, ω_c, with a phase margin PM. The system parameters are assumed known. The design procedure is best illustrated by a design example.

Consider a system with the following parameters:

K_t	Nm/A	Torque constant
$J = 2.10^{-4}$	$kg.m^2$	System moment of inertia
$R = 2$	Ω	Motor resistance
$K_a = 2$	Amp/Volt	Current amplifier gain
$N = 1000$	Counts/rev	Encoder line density

The DAC of the CDS-3310 outputs +/-10V for a 14-bit command of +/-8192 counts.

The design objective is to select the filter parameters in order to close a position loop with a crossover frequency of $\omega_c = 500$ rad/s and a phase margin of 45 degrees.

The first step is to develop a mathematical model of the system, as discussed in the previous system.

Motor

$$M(s) = P/I = K_t/Js^2 = 1000/s^2$$

Amp

$$K_a = 2 \qquad [\text{Amp/V}]$$

DAC

$$K_d = 10/32768 = .0003$$

Encoder

$K_f = 4N/2\pi = 636$

ZOH

$H(s) = 2000/(s+2000)$

Compensation Filter

$G(s) = P + sD$

The next step is to combine all the system elements, with the exception of G(s), into one function, L(s).

$L(s) = M(s)\, K_a\, K_d\, K_f\, H(s) = 3.17*10^6/[s^2(s+2000)]$

Then the open loop transfer function, A(s), is

$A(s) = L(s)\, G(s)$

Now, determine the magnitude and phase of L(s) at the frequency $\omega_c = 500$.

$L(j500) = 3.17*10^6/[(j500)^2\ (j500+2000)]$

This function has a magnitude of

$|L(j500)| = 0.00625$

and a phase

$Arg[L(j500)] = -180° - \tan^{-1}(500/2000) = -194°$

G(s) is selected so that A(s) has a crossover frequency of 500 rad/s and a phase margin of 45 degrees. This requires that

$|A(j500)| = 1$

$Arg\ [A(j500)] = -135°$

However, since

$A(s) = L(s)\, G(s)$

then it follows that G(s) must have magnitude of

$|G(j500)| = |A(j500)/L(j500)| = 160$

and a phase

$arg\ [G(j500)] = arg\ [A(j500)] - arg\ [L(j500)] = -135° + 194° = 59°$

In other words, we need to select a filter function G(s) of the form

$G(s) = P + sD$

so that at the frequency $\omega_c = 500$, the function would have a magnitude of 160 and a phase lead of 59 degrees.

These requirements may be expressed as:

$|G(j500)| = |P + (j500D)| = 160$

and

$arg\ [G(j500)] = \tan^{-1}[500D/P] = 59°$

The solution of these equations leads to:

$P = 160\cos 59° = 82.4$

$500D = 160\sin 59° = 137$

Therefore,

$D = 0.274$

and

$G = 82.4 + 0.2744s$

The function G is equivalent to a digital filter of the form:

$D(z) = 4KP + 4KD(1-z^{-1})$

where

$P = 4 * KP$

$D = 4 * KD * T$

and

$4 * KD = D/T$

Assuming a sampling period of T=1ms, the parameters of the digital filter are:

$KP = 20.6$

$KD = 68.6$

The CDS-3310 can be programmed with the instruction:

KP 20.6

KD 68.6

In a similar manner, other filters can be programmed. The procedure is simplified by the following table, which summarizes the relationship between the various filters.

Equivalent Filter Form

	CDS-3310
Digital	$D(z) = [K(z-A/z) + Cz/(z-1)] * (1-B)/(Z-B)$
Digital	$D(z) = [4\ KP + 4\ KD(1-z^{-1}) + KI/2(1-z^{-1})] * (1-B)/(Z-B)$
KP, KD, KI, PL	$K = (KP + KD) \cdot 4$
	$A = KD/(KP+KD)$
	$C = KI/2$
	$B = PL$
Continuous	$G(s) = (P + Ds + I/s) * a/S+a$
PID, T	$P = 4\ KP$
	$D = 4\ T*KD$
	$I = KI/2T$
	$a = 1/T \ln (1/PL)$

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 11 Distributed Control

Overview

In the "distributed" mode of operation, the CDS-3310 is used in conjunction with up to 7 other CDS-3310s connected via Ethernet. Programming is simplified because multiple controllers behave as a single multi-axis controller. For example, the commands PR1000,1000,1000; BGXYZ issued on the master CDS-3310 will initiate motion on three controllers. In this case, the master CDS-3310 controls the X axis, the first slave CDS-3310 controls Y, and the second slave CDS-3310 controls Z. Only the master controller requires an application program and/or the host PC communicates only with the master controller. The following commands are used for distributed control: HA, HC, HQ, HW, ZA, and ZB. **DISTRIBUTED CONTROL MUST BE PERFORMED ON A CLOSED MACHINE NETWORK.**

Configuring Distributed Control

The HA command establishes the axis order (YZWEFGH) of the slave controllers by controller serial number and the HC command configures all the slaves in a particular system. HC uses the BOOTP packets from the slaves to automatically select IP addresses and set up the system.

The HC command indicates the number of axes (including the master), update rate, and type of communication. The data update rate specifies the rate at which each slave sends a data packet to the master containing current status information. These "slave data records" are used by the master controller to make decisions based on the status of the slave controllers.

The data contained in the record is as follows:

reference position
encoder position
position error
velocity
torque
limit and home switches
axis status (in motion, motor off, at speed, stopcode)
uncommitted inputs
uncommitted outputs
two user-defined variables (ZA and ZB)

In order for the HC command to be initiated, an IP address must first be assigned to the master (see chapter 2). The slaves will typically remain without IP addresses.

Once HC is initiated, the master controller will listen for BOOTP packets from the slave controllers without IP addresses. As it receives these packets, the master will configure the slave axes according to the HA command. Once this connection has been established, the master will initiate the slave data records.

The full procedure for this method is as follows:

Step 1. Assign IP address to master controller through SmartTerm (see Chapter 2)

Step 2. Determine serial numbers of the slave controllers with the Find Ethernet Controllers feature in SmartTerm.

Step 3. Issue the HA command indicating which slaves correspond to which axes in the system, where the arguments to HA are the serial numbers found in step 2.

Step 4. Issue the command HCn,m,o where n is the total number of axes, m is the slave data update rate in milliseconds, and o is a 1 for UDP communication or 2 for TCP/IP communication. When using UDP communication, the HC command will assign one handle for both commands and slave data records. When using TCP/IP communication, the HC command will assign one handle for commands and one handle for slave data records. If o is a 3, then TCP/IP is used for commands, and UDP is used for slave data records. A total of eight handles are available.

Step 5. Poll the operand _HC for success of connection. A response of 1 indicates the command is currently executing, a 2 is for a successful configuration, and a 0 is for a failed configuration or no HC issued.

NOTE: The HC command may take up to 20 seconds to complete due to the time involved in waiting for the BOOTP packets.

Configuration Example

The example below shows a typical setup file for a three axis (three CDS-3310s) distributed control system. This example is for a UDP system, with one handle used per slave. The IP addresses of the slaves are unassigned but the IP address of the master needs to have been assigned. The HC command will automatically assign IP addresses to the slaves.

Instruction	Interpretation
#AUTO	;'This program runs on the master controller when it powers up
HA 5522,5533	;'The controller with serial number 5522 will be Y and the 'controller with serial number 5533 will be Z.
HC 3,20,1	;'3 axis system, 20 msec update rate, UDP connection
#LOOP; JP#LOOP,_HC=1	;'Wait while automatic configuration operates. This could take 'up to 10+ seconds.
IF (_HC=0)	;'Test for HC success. 0 = failed while 2 = success.
MG"CONFIGURATION FAILED"	
ELSE	
MG"CONFIG SUCCESS"	
ENDIF	
EN	

Manual Slave IP Configuration

It may be desired to manually assign IP addresses to the slaves, while still using the HC command to connect to these slaves. This is possible, but you will need to take into account the addressing scheme the HC command uses. When the HC command is initiated, the master will ARP addresses where it expects slave controllers to reside. If no controllers respond to the ARPs, the master will then listen for the BOOTP packets from un-assigned slave controllers.

For addressing the slaves manually, the IP address MUST be assigned as follows. This will insure that the HC command will properly configure these controllers based on the master IP address.

Assume Master IP address = m.n.o.p where m, n, o and p is a valid Ethernet IP address.

First Slave IP address (Axis B) = m.n.o.p+2

Second Slave IP address (Axis C) = m.n.o.p+4

…

For example, in a 4 axis system the following IP addresses would be set:

Master (X Axis) IP address – 10.10.50.10

B Axis CDS-3310 – 10.10.50.12

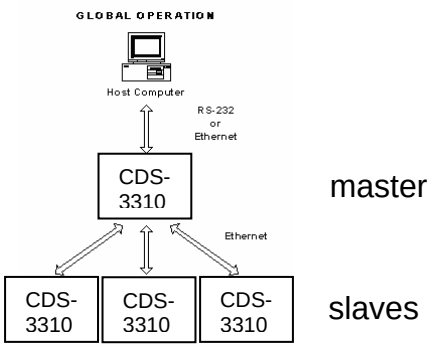
C Axis CDS-3310 – 10.10.50.14

D Axis CDS-3310 – 10.10.50.16

Global versus Local Commands

Global Commands

The command set of the controller is divided into global and local commands. When a global command is sent to the master CDS-3310 (e.g. from a PC or an application program running on the master), it affects one or more controllers in the system (axes A to H) just as if the master were a multi-axis controller.



For example, KP 10,,20 sets the P gain on the master (X) and the second slave (Z) and SP*=1000 sets the speed on all controllers. The list of global commands is below. All commands not in the table are local.

Global Command Listing

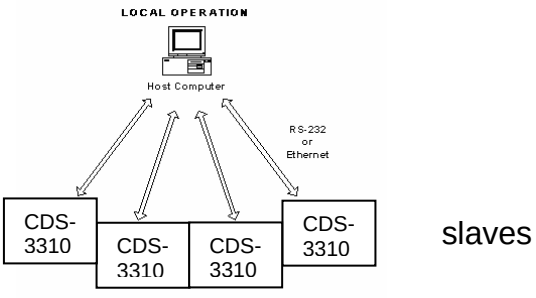
AB	AS	DC	IL	MF	@OUT	SC	TT
AC	AU	DE	@IN	MO	PA	SH	TV
AD	AW	DP	IP	MR	PL	SP	TW
AF	BG	DV	IT	MT	PR	ST	TZ
AG	BL	ER	JG	NB	PT	TD	WC
AL	BR	FA	KD	NF	QH	TE	
AM	BW	FE	KI	NZ	RC	TI	
@AN	CB	FI	KP	OB	RD	TK	
AO	CD	FL	_LF*	OC	RL	TL	
AP	CE	FV	_LR*	OE	RP	TP	
AR	CM	HM	MC	OF	SB	TS	

Local Commands

Local commands only affect a single controller and take one of three forms:

(1) A local command is sent to the master. Any command (such as BN) not in the table above will only affect the master.

(2) An application program runs on a slave controller or a PC sends a command to a slave. The command will affect only that slave. Any motion commands must reference the X axis (e.g. BGX).



(3) An SA command is sent to a slave. For the master controller to send a command that is not on the global list to a slave, it must send an SA command.

Accessing the I/O of the Slaves

The I/O of the slaves is settable and readable from the master. The bit numbers are adjusted by the handle number of the slave controller (use TH on the master to determine which slave is connected to which handle). Each handle adds 100 to the bit number. Handle A is 100 and handle H is 800. In a TCP/IP control setup with two handles per slave, it is imperative that you send commands to the first handle designated as the "command" handle.

Slave I/O may be set globally according to the following numbering scheme:

GlobalBit = (Slave Handle * 100) + LocalBit.

For example, to set digital output bit 2 on a UDP distributed slave using the E handle for communication, the E handle would have a numerical value of 500 and the bit has a value of 2. The command would therefore be SB 502.

The command TZ can be used to display all of the digital I/O contained in a distributed control system. Specific slave controllers may be queried by issuing TZn where n is the specific Ethernet handle.

The table below lists global commands that can be issued on the master to query slave I/O:

I/O Type	**Global Commands** (Handle * 100) + bit
Digital Inputs	@IN, TI
Digital Outputs	SB, CB, @OUT
Analog Inputs	@AN
Analog Outputs	AO

Appendices

Electrical Specifications

Servo Control

A+,A-,B+,B-,IDX+,IDX- Encoder and Auxiliary Encoder	TTL compatible, but can accept up to +/-12 Volts. Quadrature phase on CHA,CHB. Can accept single-ended (A+,B+ only) or differential (A+,A-,B+,B-). Maximum A,B edge rate: 12 MHz. Minimum IDX pulse width: 80 nsec.

Input / Output

Limit Switch Inputs, Home Inputs.	TTL (0-5 Volts). Pulled up internally to 5VDC through 4.7kOhm resistor. Ground input to change state.
IN[1] thru IN[8] Uncommitted Inputs and Abort Input	
OUT[1] thru OUT[10] Outputs:	TTL (0-5Volts)
IN[81], IN[82]	Auxiliary Encoder Inputs for A (X) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.

Power

18 to 72 VDC

Performance Specifications

Minimum Servo Loop Update Time	250 μsec (1 ms distributed)
Position Accuracy:	+/-1 quadrature count
Velocity Accuracy:	
Long Term	Phase-locked, better than 0.005%
Short Term	System dependent
Position Range:	+/-2147483647 counts per move
Velocity Range:	Up to 12,000,000 counts/sec servo
Velocity Resolution:	2 counts/sec
Motor Command Resolution:	16 bit or 0.0003 V
Variable Range:	+/-2 billion
Variable Resolution:	$1 \cdot 10^{-4}$
Array Size:	8000 elements, 30 arrays
Program Size:	1000 lines x 80 characters

Connectors for CDS-3310

J1 MOTOR Output 4-pin AMP Universal Mate-N-Lok

1 NC
2 A
3 C
4 B

J2 ENCODER 15-PIN Hi-density Female D-sub

1 I+
2 B+
3 A+
4 AB+
5 Ground
6 I-
7 B-
8 A-
9 AA-
10 Hall A
11 AA+
12 AB-
13 Hall B
14 Hall C
15 5 V

J3 I/O 37-pin Female D-sub

1 Reset*	20 Analog Ground
2 Error Output*/Amp Enable	21 Output 4
3 Output 3	22 Output 2
4 Output 1	23 Encoder-compare output
5 Analog input 1 (0V – 5V)	24 Analog input 2 (0V – 5V)
6 Input 7	25 Input 6
7 Input 5	26 Input 4
8 Input 3	27 Input 2
9 Input 1	28 Forward limit
10 5 V	29 Reverse limit
11 Ground	30 Home
12 +12 V	31 -12 V
13 Ground	32 Brake output (500mA max)
14 Brake Power	33 Input 8+
15 Input 8-	34 Output 10
16 Output 9	35 Output 8
17 Output 7	36 Output 6
18 Output 5	37 ABORT*
19 Analog output (16 bit resolution ±10V)	

*Active Low

J4 RS-232 Serial Port

Standard connector and cable, 9Pin Male D-sub

Pin	Signal
1	CTS – output
2	Transmit data-output
3	Receive data-input
4	RTS – input
5	Gnd
6	CTS – output
7	RTS – input
8	CTS – output
9	NC (can connect to +5V or sample clock)

J5 POWER 2-pin AMP Universal Mate-N-Lok

1 +VDC (18 to 72V)
2 GND

J8 100Base-T Ethernet

100 BASE-T/10 BASE-T - Kycon GS-NS-88-3.5

Pin	Signal
1	TXP
2	TXN
3	RXP
4	NC
5	NC
6	RXN
7	NC
8	NC

Mating Connectors

Location	Mating Connector	Terminal Pins
J5 POWER (2-pin)	AMP# 172165-1	AMP# 170361-1
J1 MOTOR (4-pin)	AMP# 172167-1	AMP# 170361-1

Pin-Out Description for CDS-3310

Outputs

Amp Enable	Goes high when the amp is turned on (SH) and low when turned off (MO)
Error	The signal goes low when the position error on any axis exceeds the value specified by the error limit command, ER.
Output 1-Output 10	These 10 TTL outputs are uncommitted and may be designated by the user to toggle relays and trigger external events. The output lines are toggled by Set Bit, SB, and Clear Bit, CB, instructions. The OP instruction is used to define the state of all the bits of the Output port.

Inputs

Encoder, A+, B+	Position feedback from incremental encoder with two channels in quadrature, CHA and CHB. The encoder may be analog or TTL. Any resolution encoder may be used as long as the maximum frequency does not exceed 12,000,000 quadrature states/sec. The controller performs quadrature decoding of the encoder signals resulting in a resolution of quadrature counts (4 x encoder cycles). **NOTE:** Encoders that produce outputs in the format of pulses and direction may also be used by inputting the pulses into CHA and direction into Channel B and using the CE command to configure this mode.
Encoder Index, I+	Once-Per-Revolution encoder pulse. Used in Homing sequence or Find Index command to define home on an encoder index.
Encoder, A-, B-, I-	Differential inputs from encoder. May be input along with CHA, CHB for noise immunity of encoder signals. The CHA- and CHB- inputs are optional.
Auxiliary Encoder, Aux A+, Aux B+, Aux I+, Aux A-, Aux B-, Aux I-	Inputs for additional encoder. Used when an encoder on both the motor and the load is required. Not available on axes configured for step motors.
Abort	A low input stops commanded motion instantly without a controlled deceleration. Also aborts motion program.
Reset	A low input resets the state of the processor to its power-on condition. The previously saved state of the controller, along with parameter values, and saved sequences are restored.
Forward Limit Switch	When active, inhibits motion in forward direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Reverse Limit Switch	When active, inhibits motion in reverse direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.

Home Switch	Input for Homing (HM) and Find Edge (FE) instructions. Upon BG following HM or FE, the motor accelerates to slew speed. A transition on this input will cause the motor to decelerate to a stop. The polarity of the Home Switch may be set with the CN command.
Input 1 - Input 8	Uncommitted inputs. May be defined by the user to trigger events. Inputs are checked with the Conditional Jump instruction and After Input instruction or Input Interrupt. Input 1 can be used for the high-speed input latch (AL and RL commands).
Latch	High speed position latch to capture axis position within 20 nano seconds on occurrence of latch signal. AL command arms latch. Input 1 is latch.

Jumper Description for CDS-3310

Label	Function (If jumpered)
10B (Rev B and up)	When OFF, the controller will use the auto-negotiate function to set the Ethernet connection speed to either 10 or 100 Base-T. When the DIP switch is ON, the controller defaults to 10BaseT.
MRST	Master Reset enable. Returns controller to factory default settings and erases EEPROM. Requires power-on or RESET to be activated.
UPGD	Used to upgrade controller firmware when resident firmware is corrupt.
9600	Sets serial port baud rate to 9600 baud when installed. Baud rate is 19200 baud when no jumper is installed.
MO	Controller will force motor to Motor Off state upon Reset
OPT	Reserved

Accessories and Options

PART NUMBER	DESCRIPTION
CDS-3310	1-axis motion controller with 500W servo drive; Ethernet/RS232
-DIN	DIN-rail mounting clip
ICM-3300	Interconnect module with opto-isolation and screw terminals
DB-28040	I/O expansion board for 8 analog inputs and 40 digital I/O (outputs source 3.3 V)
DB-28040-5V	DB-28040 with 5 V digital outputs (maximum 24)
SR-19900	Shunt regulator for CDS-3310
CABLE-Ethernet	Ethernet cables. Consult factory
CABLE-9-pinD	RS232 cable
Cable-15-1m	15-pin High Density D to discrete wires in 1 meter length (-2m for 2 meter length)
Galil CD-ROM:	
DMC SmartTERM	Smart Terminal Window with Program Editor and Syntax Help
WSDK-32	Servo Design Kit for Windows 95 / 98 / NT (Password Required)
DMCWIN32	Windows API Tool Kit (VB, C, C++, etc.) Included with Utilities
ActiveX Tool Kit	Visual Basic™ Tool Kit (includes VBXs and OCXs) (Password Required)

CAD-to-DMC	AutoCAD® DXF translator (Password Required)
MCS	Motion Control Selector. Utility for motor / amplifier sizing.
HPGL	HPGL translator (Password Required)

ICM-3300

The ICM-3300 Interconnect module breaks out the 15 pin and 37 pin connectors on the CDS-3310 into screw-type terminals to allow for easier wiring of external devices. It also provides opto-isolation for all digital I/O except the following: brake output, output compare, reset input and digital input 8. Additionally, it converts four of the CDS-3310's standard digital outputs into high-power (500mA @ 24V) sourcing outputs.

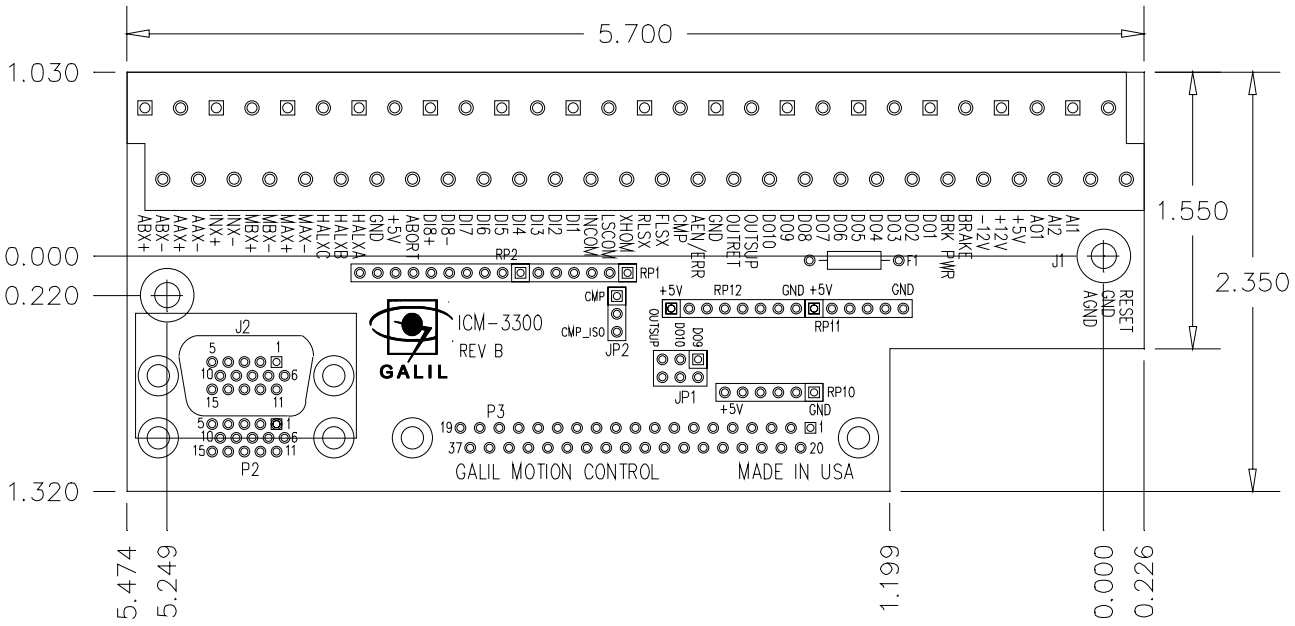


Figure 1: ICM-3300 Layout and Dimensions

Opto Isolation Settings

The ICM-3300 module allows for opto-isolation on digital inputs and outputs. This includes the dedicated I/O including limits, homes, and abort. The limits and home are powered by Limit Switch Common. Digital inputs 1-7 and the Abort Input are powered by Input Common. The digital outputs are also optically isolated including the Error and Amp-Enable outputs. The first four outputs are configured for hi-side drive only (sourcing). The supply voltage must be connected to *output supply voltage*, and the supply return must be connected to *output return.*

Input Isolation

Opto-isolation of the general purpose inputs and the abort input is used by powering the Input Common line. The limit switch and home inputs are powered by Limit Switch Common. Shown below is the circuit diagram for the isolated inputs.

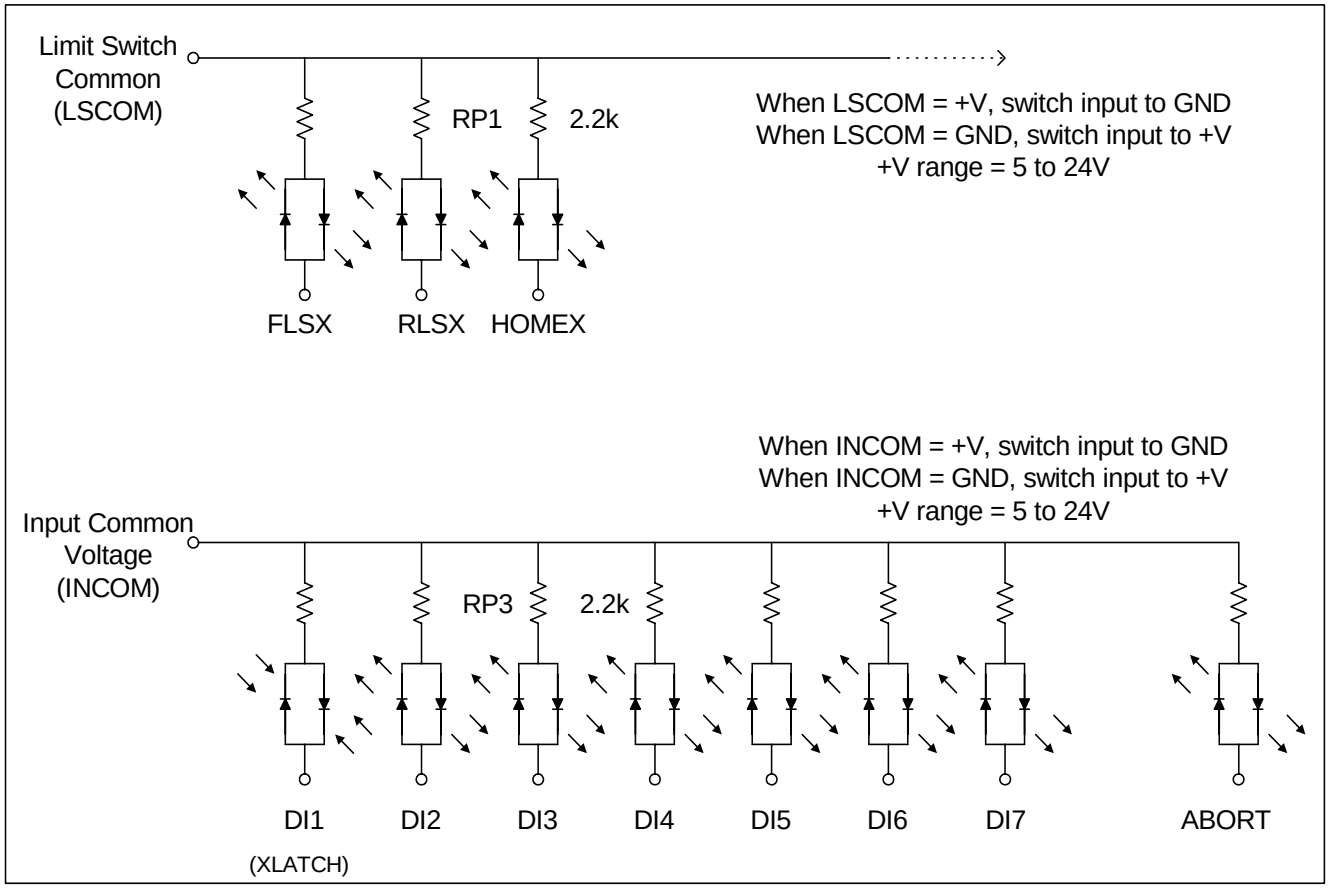


Figure 2: ICM-3300 Digital Input Isolation

Output Isolation

The first four digital outputs on the ICM-3300 are set up as high current isolated outputs and are configured for High Side (Sourcing) operation. The outputs are capable of sourcing 500 mA per output. Digital outputs 5-10 and the Amp Enable/Error output are low power opto-outputs. The output will be at the voltage that is supplied to the OUTSUP pin. Up to 24 VDC may be supplied to OUTSUP.

The resistor packs RP10, RP11 and RP12 allow for the configuration of the active state of the outputs. For example when you issue the SB1 command, the polarity of the resistor pack will determine whether the output is turned on or off.

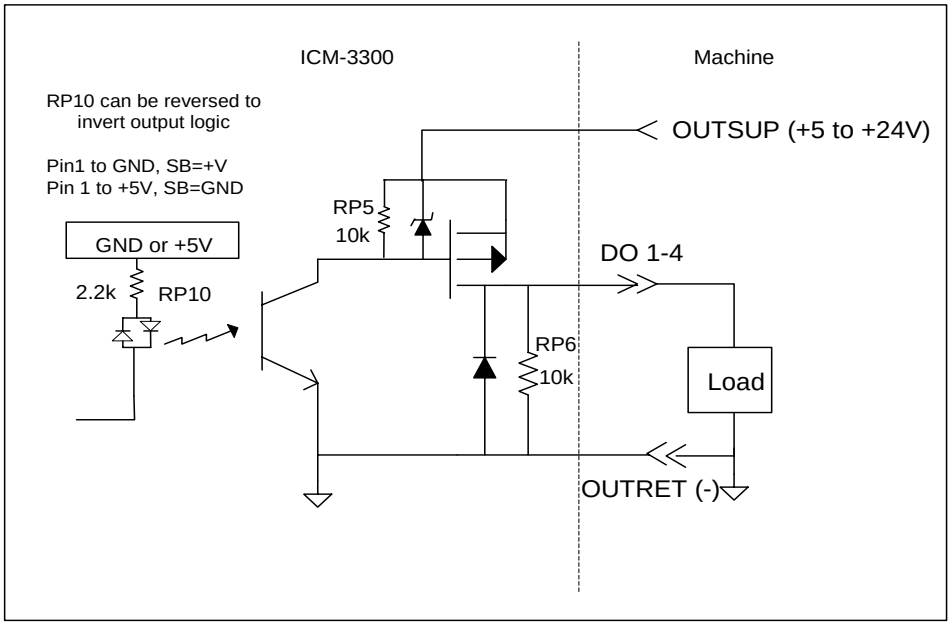


Figure 3: ICM-3300 General-Purpose Opto-Isolation for Outputs 1-4

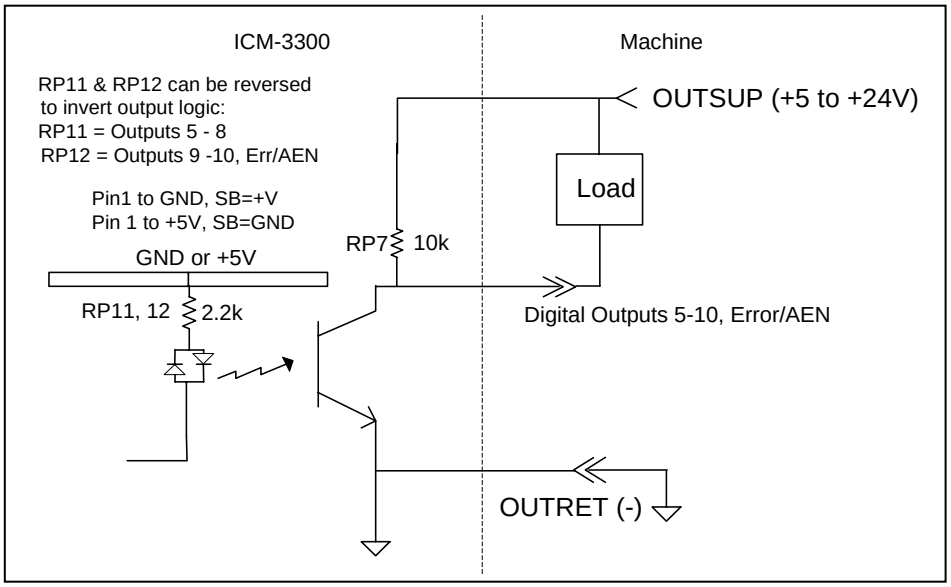


Figure 4: Outputs 5-10 and Error/Amp Enable are low-power opto-outputs with the above circuit

Screw Terminal Description

Terminal	Silkscreen	Description	Opto	In/Out
1	ABX+	Aux. Encoder B+		I
2	ABX-	Aux. Encoder B-		I
3	AAX+	Aux. Encoder A+		I
4	AAX-	Aux. Encoder A-		I
5	INX+	Main Encoder Index +		I
6	INX-	Main Encoder Index -		I
7	MBX+	Main Encoder B+		I
8	MBX-	Main Encoder B-		I
9	MAX+	Main Encoder A+		I
10	MAX-	Main Encoder A-		I
11	HALXC	Hall C		I
12	HALXB	Hall B		I
13	HALXA	Hall A		I
14	GND	Ground		O
15	+5V	5V		O
16	ABORT	Abort Input	✓	I
17	DI8+	Digital Input 8+		I
18	DI8-	Digital Input 8-		I
19	DI7	Digital Input 7	✓	I
20	DI6	Digital Input 6	✓	I
21	DI5	Digital Input 5	✓	I
22	DI4	Digital Input 4	✓	I
23	DI3	Digital Input 3	✓	I
24	DI2	Digital Input 2	✓	I
25	DI1	Digital Input 1	✓	I
26	INCOM	Input Common		I
27	LSCOM	Limit Switch Common		I
28	XHOM	Home Input	✓	I
29	RLSX	Reverse Limit Input	✓	I
30	FLSX	Forward Limit Input	✓	I
31	CMP	Output Compare		O
32	AEN/ERR	Amp Enable/Error	✓	O
33	GND	Ground		O
34	OUTRET	Output Power Return		I
35	OUTSUP	Output Power Supply		I
36	DO10	Digital Output 10	✓	O
37	DO9	Digital Output 9	✓	O
38	DO8	Digital Output 8	✓	O
39	DO7	Digital Output 7	✓	O
40	DO6	Digital Output 6	✓	O
41	DO5	Digital Output 5	✓	O
42	DO4	Digital Output 4	✓	O
43	DO3	Digital Output 3	✓	O
44	DO2	Digital Output 2	✓	O
45	DO1	Digital Output 1	✓	O
46	BRK PWR	Brake Power Supply		I
47	BRAKE	Brake Output (Sinking)		O
48	-12V	-12V output		O
49	+12V	+12V Output		O
50	+5V	+5V Output		O
51	AO1	Analog Output 1		O
52	AI2	Analog Input 2		I
53	AI1	Analog Input 1		I
54	AGND	Analog Input Ground		O
55	AGND	Analog Ground		O
56	RESET	Reset Input		I

SR-19900

Introduction

For applications requiring a shunt regulator, Galil offers a small mountable model that can be configured for varying voltage levels. Two fixed voltage threshold settings are available with jumpers, which can be set at either 33 or 66 volts. Additionally, a user defined voltage threshold can be set by changing a simple resistor. This shunt regulator operates with hysteresis, where the regulator switches on at the set voltage threshold and switches off at 2 volts below.

The shunt regulator should be placed in parallel with the power supply as in the figure below, and it should be mounted to a metal surface using thermal grease to aid in heat transfer. Connections are made to the unit at VS (voltage supply) and PG (power ground) using either the 4-pin Molex connector or the 8-pin Mate 'N Lock connector (AMP# 770579-1).

For a summary of shunt regulator operation, as well as details to help determine if one is required in your system, please refer to application note #5448 at: (http://www.galilmc.com/support/appnotes/miscellaneous/note5448.pdf).

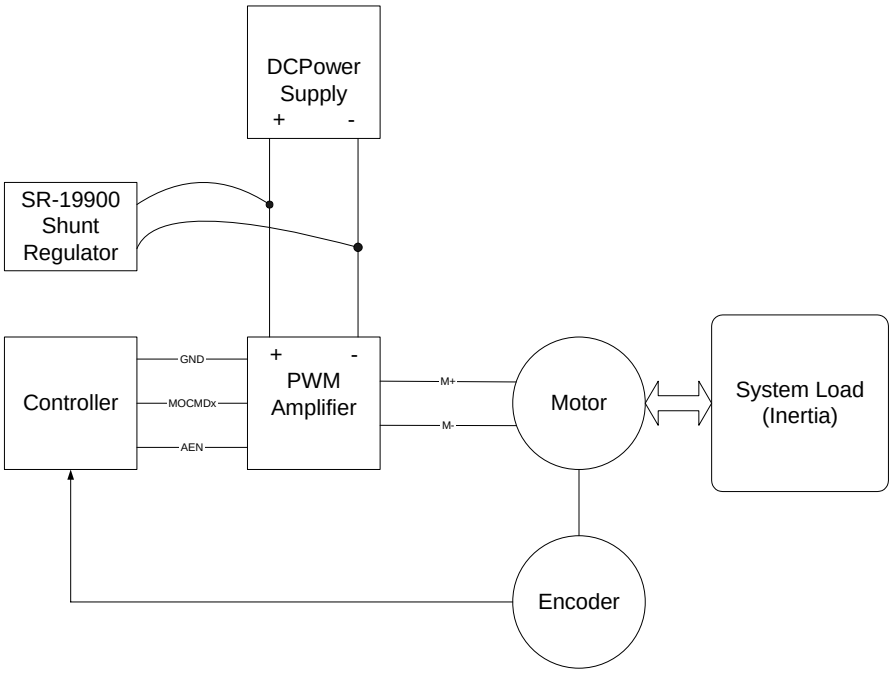


Figure 1 Shunt Regulator Placement in a Typical Servo System

Layout

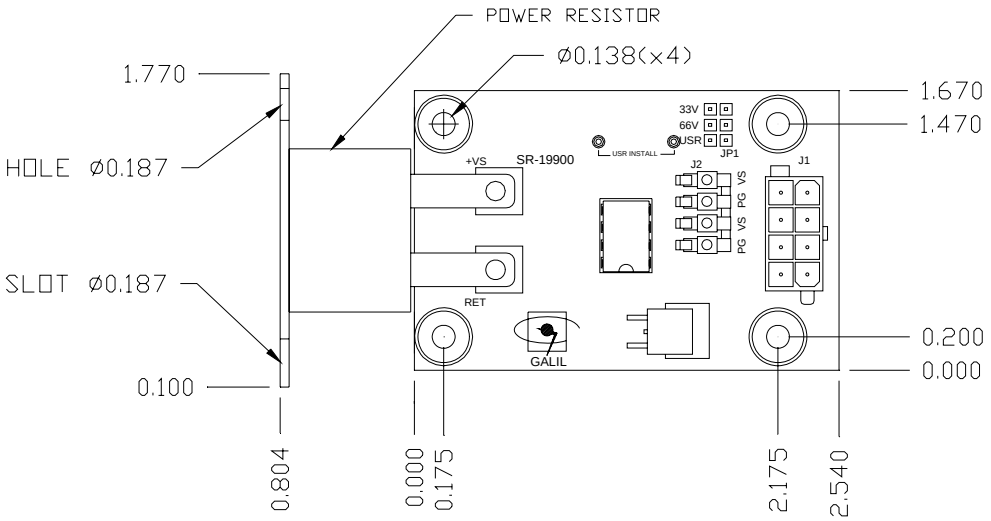


Figure 2 SR-19900 Layout

Pinout

	J2 4-pin Molex
Power Ground	**1** PG
Voltage Supply	**2** VS
Power Ground	**3** PG
Voltage Supply	**4** VS

J1 8-pin Mate 'N Lock	
1 Earth	**5** PG
2 VS	**6** PG
3 VS	**7** PG
4 VS	**8** PG

Configuration

SR-19900 Configuration	
Voltage Threshold Setting (Vs)	**JP1**
33 volts	33V
66 volts	66V
User selectable	USR

USR - User Settable Voltage R8 = 1930 * Vs -42.2K	
Voltage (Vs)	**R8 value (ohms)**
24	4.12 k
48	50.44 k
72	96.76 k

DB-28040

Introduction

The DB-28040 is an I/O expansion daughter board for the CDS-3310 motion controller that provides 8 analog inputs and 40 digital I/O.

Electrical Specifications

The analog input impedance is 45.7k. Standard configuration allows for 3.3 V digital I/O. The DB-28040-5V option provides up to 24 Open-collector outputs that can sink to 5 Volts (3mA). In the -5V configuration, the first 3 banks of 8 bits may be configured as outputs. There is no limitation on the placement of inputs. The following tables illustrate the various Input/Output combinations:

DB-28040 Valid I/O	
Inputs	Outputs
40	0
32	8
24	16
16	24
8	32
0	40

DB-28040-5V Valid I/O	
Inputs	*Outputs
40	0
32	8
24	16
16	24

***Note: Only the first three I/O banks may be configured as outputs**

16 bit Analog Inputs

The DB-28040 comes with eight 12-bit analog inputs standard. For 16-bit analog inputs, order the DB-28040-16. When using the AF command with the -16 version, the controller requires the 16-bit Analog Feedback firmware.

Interfacing to the Digital I/O

The DB-28040-5V offers up to 32 digital I/O points that can interface directly to I/O racks such as Grayhill 70GRCM32-HL, Galil IOM-1964, and OPTO-22 (see next section). These I/O points can be configured as inputs or outputs in 8 bit increments with the CO command. The I/O points accessed from the J1 header of the DB-28040-5V are connected directly to the I/O rack using a 50-pin IDC ribbon connector. Unused banks must be configured as inputs and will be grounded by the I/O rack.

Using Opto-22 G4 Series Racks with DB-28040

When using Opto-22 G4 series I/O racks with Galil extended I/O, special considerations must be made depending on the models described below. Due to changes to the pinout orientation of the 50-pin header of the G4 rack input, damage may occur to Galil components, or the opto rack may not be properly powered.

In the past, Opto-22 racks adhered to a “standard” pinout configuration for the 50-pin IDC header that mated with Galil extended I/O headers. The standard pinout configuration of the 50-pin header on Galil products allows the Galil to power the opto-isolated relays by supplying 5 VDC at pin-49.

G4PB8, G4PB16, G4PB24

The 50-pin headers on these opto racks do not connect pin-49 to the supply input for the modules. The user will have to supply 5 VDC to the terminal marked LOGIC (+) on the rack. This can be accomplished by simply running a single lead from the Galil 5 V supply to LOGIC (+) [note: the (-) return is already connected by the header].

G4PB8H, G4PB16H

The 50-pin headers on these racks actually connect both pin-49 and pin-1 to the logic supply. The user does not have to supply 5 V; however, pin-1 of the ribbon cable MUST be cut to avoid damage. Pin-1 leads to an actual unused I/O channel on the Galil and damage will occur to the extended I/O board if the 5 V line is left connected.

G4PB32H, G4PB32DEC

These 32 channel opto racks are NOT compatible with Galil extended I/O headers. Damage may occur if the connection is made with Galil extended I/O headers. The pinout configurations are laid out for a specific DEC computer interface and PAMUX B4 brain boards.

Pinout

J1 – 40 Digital I/O (50-Pin IDC Header)	
1 Bank 4 – Bit 40	2 Bank 5 – Bit 41
3 Bank 4 – Bit 39	4 Bank 5 – Bit 42
5 Bank 4 – Bit 38	6 Bank 5 – Bit 43
7 Bank 4 – Bit 37	8 Bank 5 – Bit 44
9 Bank 4 – Bit 36	10 Bank 5 – Bit 45
11 Bank 4 – Bit 35	12 Bank 5 – Bit 46
13 Bank 4 – Bit 34	14 Bank 5 – Bit 47
15 Bank 4 – Bit 33	16 Bank 5 – Bit 48
17 Bank 3 – Bit 32	18 Bank 6 – Bit 49
19 Bank 3 – Bit 31	20 Bank 6 – Bit 50
21 Bank 3 – Bit 30	22 Bank 6 – Bit 51
23 Bank 3 – Bit 29	24 Bank 6 – Bit 52
25 Bank 3 – Bit 28	26 Bank 6 – Bit 53
27 Bank 3 – Bit 27	28 Bank 6 – Bit 54
29 Bank 3 – Bit 26	30 Bank 6 – Bit 55
31 Bank 3 – Bit 25	32 Bank 6 – Bit 56
33 Bank 2 – Bit 24	34 GND
35 Bank 2 – Bit 23	36 GND
37 Bank 2 – Bit 22	38 GND
39 Bank 2 – Bit 21	40 GND
41 Bank 2 – Bit 20	42 GND
43 Bank 2 – Bit 19	44 GND
45 Bank 2 – Bit 18	46 GND
47 Bank 2 – Bit 17	48 GND
49 5 V	50 GND

J3 – 8 Analog Inputs (16-Pin IDC Header)	
1 GND	2 GND
3 Analog Input 1	4 Analog Input 2
5 Analog Input 3	6 Analog Input 4
7 Analog Input 5	8 Analog Input 6
9 Analog Input 7	10 Analog Input 8
11 GND	12 GND
13 -12 Volt Supply	14 +12 Volt Supply
15 5 V	16 GND

Layout

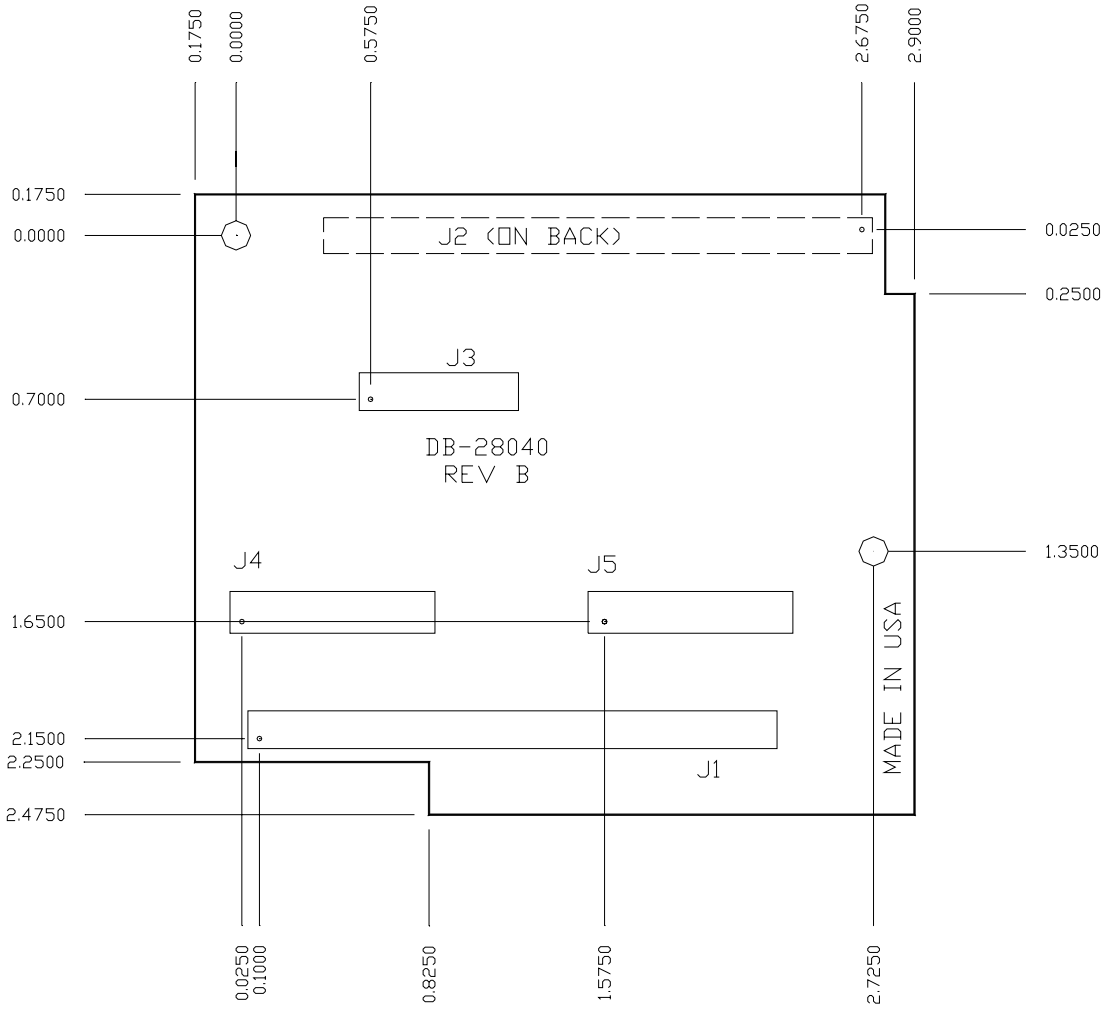


Figure 3 DB-28040 Interconnect Mounting Dimensions (Overall Dimensions: 3.075" x 2.650")

CABLE-15-1m

The **Cable-15-1m** is a cable with a 15 pin high-density connector at one end and flying leads at the other that is used to break out the J2 connector on the CDS-3310. This 15 pin cable provides an interface to the encoder and hall sensors. This section lists the color coding for 15 pin cable. (2 meter cables are also available with -2m designation).

15 Pin Break-out Cable (for J2) Color Coding

Pin	*Color*	Pin	*Color*
1	Black	**9**	Green
2	Black/White	**10**	Blue
3	Brown	**11**	Purple
4	Brown/White	**12**	Silver
5	Red	**13**	White
6	Red/White	**14**	Pink
7	Orange	**15**	Light Green
8	Yellow		

Communicating with OPTO-22 SNAP-B3000-ENET

The controller is connected to OPTO-22 via handle F. The OPTO-22's IP address is 131.29.50.30. The Rack has the following configuration:

Digital Inputs	Module 1
Digital Outputs	Module 2
Analog Outputs (+/-10V)	Module 3
Analog Inputs (+/-10V)	Module 4

Instruction	**Interpretation**
#CONFIG	Label
IHF=131,29,50,30<502>2	Establish connection
WT10	Wait 10 milliseconds
JP #CFGERR,_IHF2=0	Jump to subroutine
JS #CFGDOUT	Configure digital outputs
JS #CFGAOUT	Configure analog outputs
JS #CFGAIN	Configure analog inputs
MBF = 6,6,1025,1	Save configuration to OPTO-22
EN	End
#CFGDOUT	Label
MODULE=2	Set variable
CFGVALUE=$180	Set variable
NUMOFIO=4	Set variable
JP #CFGJOIN	Jump to subroutine

#CFGAOUT	Label
MODULE=3	Set variable
CFGVALUE=$A7	Set variable
NUMOFIO=2	Set variable
JP #CFGJOIN	Jump to subroutine
#CFGAIN	Label
MODULE=5	Set variable
CFGVALUE=12	Set variable
NUMOFIO=2	Set variable
JP#CFGJOIN	Jump to subroutine
#CFGJOIN	Label
DM A[8]	Dimension array
I=0	Set variable
#CFGLOOP	Loop subroutine
A[I]=0	Set array element
I=I+1	Increment
A[I]=CFGVALUE	Set array element
I=I+1	Increment
JP #CFGLOOP,I<(2*NUMOFIO)	Conditional statement
MBF=6,16,632+(MODULE*8),NUMOFIO*2,A[]	Configure I/O using Modbus function code 16 where the starting register is 632+(MODULE*8), number of registers is NUMOFIO*2 and A[] contains the data.
EN	end
#CFGERR	Label
MG"UNABLE TO ESTABLISH CONNECTION"	Message
EN	End

Using the equation

I/O number = (Handlenum*1000) + ((Module-1)*4) + (Bitnum-1)

MG @IN[6001] display level of input at handle 6, module 1, bit 2

SB 6006 set bit of output at handle 6, module 2, bit 3 to one

or

OB 6006,1

AO 608,3.6 set analog output at handle 6, module 53, bit 1 to 3.6 volts

MG @AN[6017] display voltage value of analog input at handle6, module 5, bit 2

List of Other Publications

"Step by Step Design of Motion Control Systems"

by Dr. Jacob Tal

"Motion Control Applications"

by Dr. Jacob Tal

"Motion Control by Microprocessors"

by Dr. Jacob Tal

Training Seminars

Galil, a leader in motion control with over 500,000 controllers working worldwide, has a proud reputation for anticipating and setting the trends in motion control. Galil understands your need to keep abreast with these trends in order to remain resourceful and competitive. Through a series of seminars and workshops held over the past 20 years, Galil has actively shared their market insights in a no-nonsense way for a world of engineers on the move. In fact, over 10,000 engineers have attended Galil seminars. The tradition continues with three different seminars, each designed for your particular skill set-from beginner to the most advanced.

MOTION CONTROL MADE EASY

WHO SHOULD ATTEND

Those who need a basic introduction or refresher on how to successfully implement servo motion control systems.

TIME: 4 hours (8:30 am-12:30pm)

ADVANCED MOTION CONTROL

WHO SHOULD ATTEND

Those who consider themselves a “servo specialist” and require an in-depth knowledge of motion control systems to ensure outstanding controller performance. Also, prior completion of “Motion Control Made Easy” or equivalent is required. Analysis and design tools as well as several design examples will be provided.

TIME: 8 hours (8-5pm)

PRODUCT WORKSHOP

WHO SHOULD ATTEND

Current users of Galil motion controllers. Conducted at Galil’s headquarters in Rocklin, CA, students will gain detailed understanding about connecting systems elements, system tuning and motion programming. This is a “hands-on” seminar and students can test their application on actual hardware and review it with Galil specialists.

Attendees must have a current application and recently purchased a Galil controller to attend this course.

TIME: Two days (8:30-4:30pm)

Contacting Us

Galil Motion Control

270 Technology Way

Rocklin, California 95765

Phone: 916-626-0101

Fax: 916-626-0102

E-mail address: support@galilmc.com

URL: www.galilmc.com

FTP: www.galilmc.com/ftp

WARRANTY

All controllers manufactured by Galil Motion Control are warranted against defects in materials and workmanship for a period of 18 months after shipment. Motors, and Power supplies are warranted for 1 year. Extended warranties are available.

In the event of any defects in materials or workmanship, Galil Motion Control will, at its sole option, repair or replace the defective product covered by this warranty without charge. To obtain warranty service, the defective product must be returned within 30 days of the expiration of the applicable warranty period to Galil Motion Control, properly packaged and with transportation and insurance prepaid. We will reship at our expense only to destinations in the United States and for products within warranty.

Call Galil to receive a Return Materials Authorization (RMA) number prior to returning product to Galil.

Any defect in materials or workmanship determined by Galil Motion Control to be attributable to customer alteration, modification, negligence or misuse is not covered by this warranty.

EXCEPT AS SET FORTH ABOVE, GALIL MOTION CONTROL WILL MAKE NO WARRANTIES EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO SUCH PRODUCTS, AND SHALL NOT BE LIABLE OR RESPONSIBLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES.

COPYRIGHT (3-97)

The software code contained in this Galil product is protected by copyright and must not be reproduced or disassembled in any form without prior written consent of Galil Motion Control, Inc.

Index

Abort 21, 59, 58–60, 82, 86
  Off On Error 21
Absolute Position 15, 42, 43, 23
Absolute Value 53, 36, 59
Acceleration 20, 42, 44, 45, 11, 49
Accessories 87
Address 101
Amplifier Enable 5
Amplifier Gain 4, 72, 74
Analog Input 22, 23, 51
Analysis
  WSDK 10, 14
Arm Latch 15
Array 1, 3, 8, 42, 6, 8, 16, 20, 26, 35, 39, 40, 41, 42, 45, 46, 48, 50, 83
Automatic Subroutine
  CMDERR 19, 30
  ININT 19, 29, 30, 32, 53
  LIMSWI 20, 19, 30, 31, 59, 61, 86
  MCTIME 19, 23, 30, 32
  POSERR 19, 30, 60, 61
  TCPERR 19, 30
Automatic Subroutine XE "Subroutine"
  TCPERR 34
Auxiliary Encoder 21, 9, 10, 15, 52, 82
  Dual Encoder 40, 10
Backlash 42, 9, 10, 56
  Dual Loop 42, 9, 10, 56
Baud Rate 9, 10, 25, 65
Begin Motion 13
Binary 2, 27, 35, 37, 54
Binary Commands 38
Bit-Wise 27, 35, 45
Burn
  EEPROM 1, 3, 8, 9, 54
Capture Data
  Record 42, 8, 39, 40, 41
Clear Bit 22, 50, 86
Coordinated Motion
  Ecam 53–1, 3
  Electronic Cam 53, 2
Data Record 29, 33
Dip Switch
  Address 101
Ecam 53–1, 3
  Electronic Cam 53, 2
Echo 32, 33
Edit Mode 17, 22, 31
Editor 16, 17
EEPROM 1, 3, 8, 9, 54
Electronic Cam 53, 2
Electronic Gearing 1, 41, 50, 52
Encoder
  Auxiliary Encoder 21, 9, 10, 15, 52, 82
  Differential 4, 11, 13, 21, 52, 65, 82, 86
  Dual Encoder 40, 10
  Index Pulse 11, 20, 12
  Quadrature 3, 4, 9, 49, 55, 59, 70, 82, 83, 86
Error Code 27, 29, 39, 40, 21
Error Handling ..... ii, 18, 58
Error Limit 22, 30, 59, 86
  Off On Error 21, 60
Example
  Automatic Subroutines 32
  Communication Interrupt 44
  Contouring 6
  Cut-to-Length 42
  Design Examples 15
  Dual Loop 10
  Editor Mode 22
  Electronic Gearing 52
  General 24
  I/O Examples 50
  Input Interrupt 53
  Jog Moves 45
  JP Command 27
  Latching 15
  Limit Switch 61
  Limit Switches 31
  Motion Smoothing 11

OPTO-22 98
PF Command 47
Position Follower 51, 52
PR-PA Moves 43
Record Array 41
Sampled Dual Loop 56
Using Labels 18
Wire Cutter 55
FIFO 33
Filter Parameter
Damping 14, 65, 69, 72
Gain 14, 16, 65, 69, 70
Integrator 14, 69, 72
PID 14, 69, 70, 76
Proportional 14, 10, 69
Stability 10, 64, 65, 69, 73
Filter Parameters
Stability 57
Find Edge 20, 12, 15, 87
Formatting 45, 48, 49
Frequency 4, 72, 73, 74, 75
Function
Arithmetic 16, 27, 35, 38, 49
Gain 4, 14, 16, 65, 69, 70, 72, 74
Gear Ratio 50, 52
Gearing 1, 41, 50, 52
Halt 20, 21, 22, 23, 25, 26
Hardware 58
Address 101
I/O 50
MO Jumper 9
Home Input 20, 21, 12, 15, 39, 82
Homing 20, 12, 15, 86
Find Edge 20, 12, 15, 87
I/O
Amplifier Enable 5
Digital Input 2, 19, 36, 51
Digital Output 2, 36
Home Input 20, 12, 15, 39, 82
Limit Switch 21, 50, 18, 20, 21, 30, 31, 39, 59, 61, 65, 82, 86
ICM-1900 87
Independent Motion
Jog 1, 16, 41, 44, 45, 4
Index Pulse 11, 20, 12
ININT 19, 29, 30, 32, 53
Input Interrupt
ININT 29, 30, 32, 53
Integrator 14, 69, 72
Interconnect Module
ICM-1900 87
Internal Variable 17, 27, 37
Interrogation 14, 15, 16, 39, 40, 20, 21, 45, 47, 48
Interrupt 19, 29, 30, 32, 53
Invert 13, 20, 9, 65
Jog 1, 16, 41, 44, 45, 4, 44
Jumper 8, 9, 10, 25, 65, 87
Label 2–3
Program Label 10, 16, 18, 21, 26, 27, 29, 39
Special Label 18, 19, 23, 30, 31, 53, 60, 61
Latch 3, 32, 40, 15, 87
Arm Latch 15
Position Capture 15
Limit Switch ... 21, 50, 18, 20, 21, 30, 31, 39, 59, 61, 65, 82, 86
LIMSWI 20, 19, 30, 31, 59, 61, 86
Linear Interpolation 4
Logical Operator 26, 43
Masking
Bit-Wise 27, 35, 36, 45
Math Function 36, 40
Absolute Value 53
Sine 3
MCTIME 19, 23, 30, 32
Memory 3, 26, 35, 7, 16, 21, 26, 30, 31, 39, 40, 54
Message 28, 33, 21, 29, 36, 45, 46
Modelling 69
Motion Complete
MCTIME 19, 23, 30, 32
Motion Smoothing 42, 43, 45, 11
Motor Command 16, 72, 83
Multitasking 19, 33
OE
Off On Error 21
Off On Error 21, 32, 58–60
Operand
Internal Variable 17, 27, 37
Operator
Bit-Wise 27, 35
Output
Amplifier Enable 5, 58, 60
Digital Output .. 2, 3, 22, 20, 24, 36, 50, 53, 54, 55, 82
Error Output 22, 23, 58
Motor Command 69, 70
Output Compare 22
Output of Data 45
PID 14, 69, 70, 76
Playback 8, 39
POSERR 19, 30, 60, 61
Position Capture 15
Latch 3, 32, 40, 15, 87
Position Error . 11, 13, 14, 15, 22, 10, 19, 20, 30, 31, 40, 57, 58, 60, 68, 86
POSERR 19, 30, 60, 61
Position Limit 60
Program
Terminal 11
Program Flow 22, 51
Interrupt 18, 19, 25, 30, 32, 43, 44, 53
Stack 29, 30, 33, 53
Programming 41, 16, 65
Proportional Gain 14

Protection
Error Limit 22, 30, 59, 86
Torque Limit 11, 16
PWM 4
Quadrature 4, 9, 49, 55, 59, 70, 82, 83, 86
Quit
Abort 21, 59, 82, 86
Stop Motion 61
Record 42, 8, 39, 40, 41
Latch 3, 40, 15, 87
Teach 42, 8, 40
Register 10
Reset 5, 8, 20
Selecting Address 101
Serial Port 10, 34, 44, 46, 85
Set Bit 22, 50, 55, 86
Sine 41, 3, 36
Single-Ended 11, 13, 82
Slew 15, 20, 32, 42, 12, 23, 55, 87
Smoothing 42, 43, 45, 11
Sngle-Ended 4
Software
Terminal 8, 9, 10
WSDK 10, 14
Special Label 18, 19, 23, 30, 31, 53, 60, 61
Stability 10, 57, 64, 65, 69, 73
Stack 29, 30, 33, 53
Zero Stack 33, 53
Step Motor 86
Stop Code 40, 15, 21, 65
Stop Motion 61
subroutine 30
Subroutine. 20, 18, 19, 20, 23, 26, 29, 30, 31, 32, 34, 43, 53, 59, 60, 61, 86
Synchronization 2, 4, 53
Syntax 35, 36, 37
Tangent 36
Teach 42, 8, 40
Data Capture 40, 41
Latch 3, 32, 40, 15, 87
Play-Back 8
Record 42, 8, 39, 40, 41
Tell Code 39, 40, 22
Tell Position 15, 17, 33, 40, 38, 48
Tell Torque 40
Terminal 9, 10, 11, 20
Theory 66
Damping 14, 65, 69, 72
Digital Filter 70, 71, 74, 76
Modeling 66, 69
PID 14, 69, 70, 76
Stability 10, 57, 64, 65, 69, 73
TIME 39
Time Interval
DT 4, 5
RC 8, 40, 41
Timeout
MCTIME 19, 23, 30, 32
Torque Limit 11, 16
Trigger 16, 22, 23, 24, 26, 86, 87
Trippoint 22, 23, 24, 29
Troubleshoot 64
TTL 2, 3, 4, 19, 21, 52, 53, 58, 82, 86
Tuning 2, 14, 15, 10
Stability 10, 57, 64, 65, 69, 73
WSDK 10, 14
Upload 40
User Unit 49
Variable .. 3, 8, 17, 40, 16, 20, 21, 26, 35, 37, 38, 40, 42, 45, 46, 47, 48, 49, 83
Internal Variable 17, 27, 37
Vector Mode
Linear Interpolation 4
Tangent 36
Wire Cutter 55
WSDK 10, 14
Zero Stack 33, 53